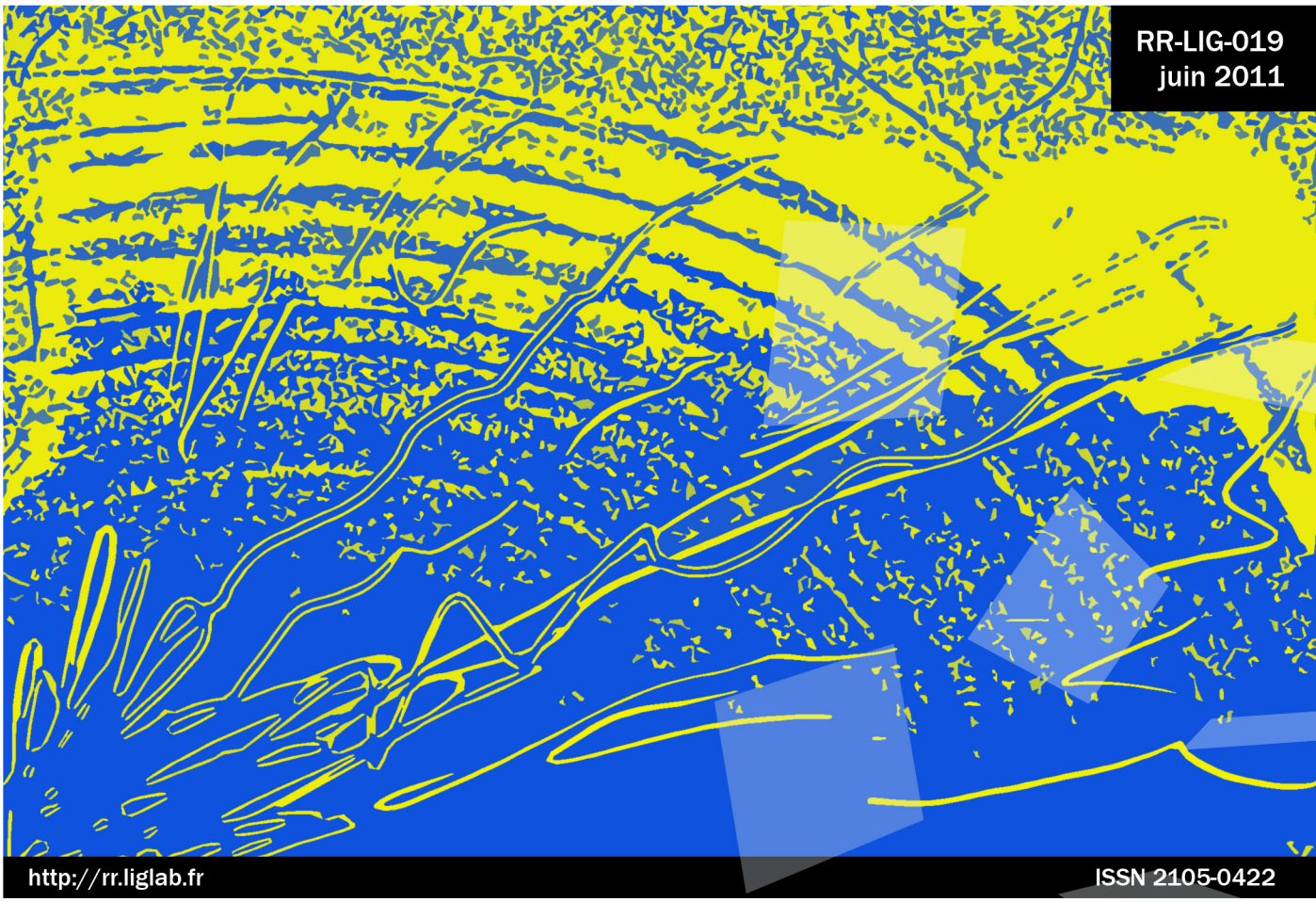
Les rapports de recherche du LIG

Astral :
An algebraic approach for sensor data stream querying

Loïc PETIT, PhD student, LIG, Orange Labs, France
Claudia Lucia RONCANCIO, Professor, LIG, Grenoble University (Grenoble-INP), France
Cyril LABBE, Associate Professor, LIG, Grenoble University (UJF), France



RR-LIG-019
juin 2011

http://rr.liglab.fr
ISSN 2105-0422

L I G

Astral: An algebraic approach for sensor data stream querying☆

Loïc Petit[a,b], Claudia Lucia Roncancio[b], Cyril Labbé[b]

[a] *Orange Labs, Meylan, France*
[b] *University of Grenoble, LIG Laboratory, France*

Abstract

The use of sensor based applications is in expansion in many contexts. Sensors are involved at several scales ranging from the individual (e.g. personal monitoring, smart homes) to regional and even world wide contexts (i.e. logistics, natural resource monitoring and forecast). Easy and efficient management of data streams produced by a large number of heterogeneous sensors is a key issue to support such applications. Numerous solutions for query processing on data streams have been proposed by the scientific community. Several query processors have been implemented and offer heterogeneous querying capabilities and semantics.

Our work is a contribution on the formalization of queries on data streams in general, and on sensor data in particular. This paper proposes the Astral algebra; defining operators on temporal relations and streams which allow the expression of a large variety of queries, both instantaneous and continuous. This proposal extends several aspects of existing results: it presents precise formal definitions of operators which are (or may be) semantically ambiguous and it demonstrates several properties of such operators. Such properties are an important result for query optimization as they are helpful in query rewriting and operator sharing. This formalization deepens the understanding of the queries and facilitates the comparison of the semantics implemented by existing systems. This is an essential step in building mediation solutions involving heterogeneous data stream processing systems. Cross system data exchange and application coupling would be facilitated.

This paper discusses existing proposals, presents the Astral algebra, several properties of the operators and the prototype we have implemented.

Keywords: formal; algebra; sensor; data stream; model; query; optimisation

1. Introduction

The expansion of sensor based applications motivates many studies of sensor data management. Several academic and industrial projects propose sensor data processing solutions including numerous proposals for query evaluation. It is difficult to compare the power of expression of different propositions because the query semantics is not always well defined. The evaluation of a query by two systems may lead to disparate results even if they work on the same sensor network. For these reasons query processing across multiple heterogeneous sensor systems is difficult and prevents extensive use of optimization techniques.

This paper is a contribution to the clarification and formalization of sensor queries. It first points out the different kind of queries on sensor data and then proposes Astral, an algebra formalizing all the related concepts and operators. Astral provides a unified model to express a large variety of sensor queries including continuous and instantaneous ones. Formal definition facilitates a better understanding of the queries and the comparison of the semantics implemented by different systems. This paper goes beyond previous work in the clarification of the semantics of important operators. Among them window operators and joins whose complexity is often underestimated. This allows us to isolate properties of the operators which can be used for query rewriting and optimization. Discussion of such properties is an important contribution of this paper. More generally, our proposal covers a large variety of queries combining streams and relations and can be helpful in mediation systems involving heterogeneous data stream processing. It would facilitate cross systems data exchange and application coupling.

This paper presents the definition of the Astral algebra☆, its implementation and a discussion of related work. It also illustrates how Astral can be used to express queries supported by several existing sensor querying systems. The paper is organized as follows: Section 2 introduces the context, related work and motivation for our proposal. Section 3 presents the core of Astral. Section 4 focus on operators on relations whereas section 5 focus on operators on streams. Section 6 presents properties of stream and relation operators. Section 7 compares Astral

☆ *A full version of this algebra is available on the Astral Wiki at* `http://sigma.imag.fr/astral`

Email addresses: `loic.petit@orange-ftgroup.com` (Loïc Petit), `Claudia.Roncancio@imag.fr` (Claudia Lucia Roncancio), `Cyril.Labbe@imag.fr` (Cyril Labbé)

to existing proposals. Section 8 presents the prototype of Astral. Section 9 gives our conclusions and research perspectives. Appendix A presents the proofs not included in the preceding sections.

2. Querying streams and models

A sensor is a device that detects or measures a physical property and records, indicates or otherwise responds to it[1]. In ubiquitous systems, a sensor mainly acts as a data stream source. Sensors may perform a local treatment on data before sending the records as streams to be further processed elsewhere. A timestamp in the record indicates when the measurement/detection has been made or computed. To allow rich declarative queries on sensor streams, any relevant property of the sensor at the time of detection may be associated to the record. These properties, referred to as meta-data, can be for example the sensor type, the sensor id, the sensor location, the accuracy of the measurement or the battery level at detection time...

For many years now, research and industrial projects [19] have focused on managing data streams in a declarative way capitalizing on the results obtained, for example, by the database community.

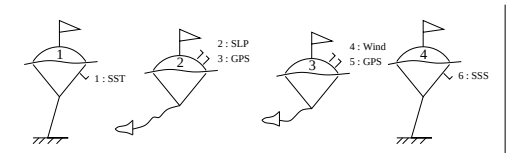
This section introduces the main approaches to query sensor data and presents the motivation for our work. It is organized as follows. Section 2.1 introduces our running example. Section 2.2 presents the variety of interesting queries to treat sensor related data (e.g., continuous queries). Section 2.3 presents related work on formal models to support such queries and gives the motivation for our work.

2.1. Running example: buoys & sensors

The main running example used in this paper involves a large set of oceanographic buoys similar to the 1250 buoys used in the Global Drifter Program [33, 31]. It includes drifting buoys and anchored buoys each with at least one sensor. Several types of sensors are used to measure Sea Surface Temperature (SST), Sea Level Pressure (SLP), Wind (W) and Sea Surface Salinity (SSS). Drifting buoys also have positioning sensors (GPS) and may have a drogue of which the depth is specified. These infrastructure and collected data serve a variety of applications on scientific calculus (based on historical data), disaster detection and tracking (real time measurement), weather forecasts (using present states and historical data),...

2.2. Of Sensors and Queries

Querying streams, and particularly sensor data, involve long living *continuous queries* where data are consumed and may not be persistent. This is different from the classic DBMS context which supports *instantaneous queries*



Buoys

BuoyId	Drift
1	0
2	1
3	1
4	0

DrifBuoys

BuoyId	DrogueD
2	20
3	15

AnchBuoys

BuoyId	Loc
1	P1
4	P4

Sensors

Id	BuoyId	Type	Unit
1	1	SST	Cel
2	2	SLP	Pa
3	2	GPS	lg/la
4	3	Wind	m/s
5	3	GPS	lg/la
6	4	SSS	ppm

Stream of Data

Id	Value	timestamp
...	...	...
4	10	1230
1	20	1410
3	45.1−5.7	1410
5	−45.1−5.7	1641
6	35.1	1853
4	10	1885
4	12	2102
3	45.2−5.4	2253
2	1024	2570
...	...	...

Figure 1: Oceanographic buoys and related data: buoys properties, sensors properties and data stream from sensors.

(the classical ones) also called "ad-hoc" queries. To cope with complete sensor querying requirements, both classes of queries are needed.

Instantaneous Queries

Definition 1 (Instantaneous query)**.** *An instantaneous query is evaluated at a time t on a set of data available at this time. The result is the set of data satisfying the query conditions.*

Queries are evaluated on data describing past or present states of the monitored system. For example: `Which is the last buoy being in zone '7016' ` [2]`?`
The answer to such a query may be found in the present (if 7016 zone is not empty) or in the past states of the zone history. Two sub-classes of instantaneous queries can be distinguished according to the data they require.

Definition 2 (Present query)**.** *A present query is an instantaneous query which is evaluated on a set of data describing the present state of the monitored system. The result is the set of data satisfying the query conditions.*

For example: `Which buoys are presently in zone '7016'?`

Definition 3 (Historic query)**.** *An historic query is an instantaneous query which is evaluated on a set of data describing past states of the monitored system. The result is the set of data satisfying the query conditions.*

For example: `What was the mean of measures issued by sensors n°42 and 44 yesterday at 20h UTC?`
The evaluation of such a query requires past data. In most cases, sensors are not able to store the history and an external persistent support is needed.

Continuous Queries

Definition 4 (Continuous query)**.** *A continuous query process without interruption a varying set of data. The result is a varying set of data satisfying the query conditions.*

[1] Oxford American dictionary

[2] see [25] for zones description

For example: `Every 30 minutes, give the mean over 30 minutes of SST measured by sensor 42`

The evaluation of this query produces a new data item every 30 minutes. The process will last until an explicit stop instruction. In this example, the input data source is a stream coming from a sensor designated by its id (42).

Among continuous queries, we highlight a particular subclass: queries by designation. These queries use a sub-query to designate data sources (here sensors/streams) to be processed by the continuous query. This allows very expressive queries but requires powerful query processing.

Definition 5 (Query by designation). *A query by designation is a query that designates its input data sources using another query.*

For example: `Every 30 min, retrieve data from SST sensors measuring more than 25°C at start time` This query will process the data issued by such sensors verifying the temperature condition when the query is launched. If, after a while, a sensor measurement drops under 25, its stream has still to be processed by the continuous query. Here, the "designation" sub-query is an instantaneous query and more precisely a present query. A continuous query can also be useful for designation as in the following example:

`Every 30 min, retrieve data from SST sensors measuring more than 25°C`

In this last query, unlike the preceding one, if after a while, a sensor measurement drops under 25, its stream has to be removed from the scope of the continuous query. On the other hand, if a sensor measurement exceeds 25, then its stream has to be added to the set of processed streams.

Several systems propose sensor data management. They respond to different constraints and implement different kinds of queries. Not all the aforementioned classes of queries are supported by each of such systems. Among the existing solutions we distinguish in-network query evaluation in sensor networks [29, 30], mainly centralized but general stream query evaluation provided by Data Stream Management Systems (DSMS) [4, 2, 1], data collectors implementing present and historic queries [10, 36, 26], and hybrid systems combining several aspects of the preceding catégories [18, 3, 15, 10].

2.3. Formal models: related work and motivation

In this paper we focus on formal support for sensor data management including both instantaneous and continuous queries. We briefly discuss in this section existing results and present the motivation for our contribution.

The early years. Data streams were explored early in the SEQ model [38] where a stream is considered as a set of records with a positional order (without time model). This formalism has been used by several works [18, 7]. Windows support and joins between two streams were not as completely analyzed as at present. In the first proposals [42], continuous queries were considered as instantaneous queries executed periodically[3]. Tables were already used as a relation that varies over time. Later on, continuous queries over full streams were considered [23]. Firstly only full or instant windows were used in these models, then real windows (sliding, fixed, tumbling) were introduced [40] following the two major criteria: time-based and count-based windows.

Network analysis and sensor monitoring motivated new models for data stream management [2, 29, 30, 45, 13]. Sliding windows, aggregation and joins were really applied. However, the semantics of the operators was mainly focused on implementation with several restrictions and merits further clarification.

The two-fold approach. One of the major contributions in data stream management is STREAM [4] and particularly its semantic model [5]. It distinguishes two major concepts: on the one hand, **streams** as infinite sets of tuples with a common schema having a timestamp, on the other hand, **relations** as functions that map time to a finite set of tuples with a common schema. In this approach, windows are mappers from stream to relation, streamers map relation to stream, and relational algebra [12] operators work on relation(s) to produce a relation. Stream to stream operations exist only as a composition of the other operators. This proposal and the associated CQL language [6] have been used ever since in many stream projects [44, 39, 22].

Rethinking formal models. Recently, the core basis of the aforementioned models has been proven to be semantically ambiguous. Therefore, standardization and clarification of the core semantics of some operators have been proposed [24]. The concept of **batch** as a set of tuples that has the same timestamp, has been introduced. A new wave of formalization arises subsequently to fill the lack of mathematical models. Some works contribute to more complete formalization of windows [32, 8, 35] as they are the aspect of stream processing which has led to the most different interpretations.

One step forward. This paper is a contribution to the formalization of sensor data management. It extends existing proposals in several ways.

- It goes a step forward in the clarification of the semantics of some operators. For instance, the window operator (range, row, slide definition) and the stream join (instantaneous, infinite, band, join) have many possible interpretations leading to different results. This has significant drawbacks in practice; without unambiguous definition of the operators it is hard to

[3] giving only new tuples to the user

mediate multiple systems. Some standardization efforts have been made [24], mainly for some common types of windows, but not every operator has been treated and more semantics can be discovered. For instance, streamers and joins have many interpretations that should be detailed.

- It improves the expressivity of some operators. For example, the power of expression of the window operator can be really large. For now, only sliding (or similar) windows in the positional or time domain were considered. This paper shows that windows can have more useful interpretations, for instance a slide by 3s every 5 tuples (combination of positional and time-based).
- It facilitates the integration of continuous, instantaneous, designation and historic queries. Current proposals hardly consider this aspect which is a main issue for future global data stream management.
- It provides concrete results on query equivalence which is a topic that received little attention until now. For example, today there are no results on how to determine if two continuous queries (even simple ones), starting at different time, can share their query plans. This paper proposes a set of properties, relying on mathematical proofs, which will allow interesting algebraic optimization.

As a matter of fact, a good comprehension of querying mechanisms is a key point for the development of ubiquitous systems. A well defined model will enhance the comprehension of requirements and allow a better reusability, coupling and development of existing supports.

On the application developer side, having in mind a specific application based on sensors, a model is needed to identify required querying capabilities. Then, technologies and systems can be chosen to fulfil such requirements.

On the infrastructure provider side, having in mind specific technologies/systems, a model is needed to represent querying capabilities. A better matching between target application requirements and systems provided functions can be made.

3. The basis of the algebra

This section introduces the foundations of the Astral algebra. For sensor stream querying, data timestamps and positions are indispensable for a correct definition of streams and relations. The following presents the definitions concerning data representation. The operators of the algebra are introduced in the following sections.

3.1. Tuples and identifiers

Before going any further, let us recall the basic principles of strict formalization of tuples in the relational model.

Definition 6 (Tuple)**.** *A tuple is a partial function from a finite subset of attribute names to atomic values. The domain of this function is called the schema of the tuple.*

As any partial function it is possible to represent a tuple as a set of couples of Attribute×Value (with a unique constraint on the attribute). We will often use this representation to manipulate tuples. Note that no type system has been defined and it is assumed that any two values can be compared. This assumption is not a problem from a programming language theory point of view.

Now let us add a physical identifier to tuples. This particular attribute is intended to identify tuples even when the values of the other attributes are duplicated. It also allows the definition of a positional order in a set of tuples. We now consider the definition of physical identifier, tuple-set and the position of a tuple.

Definition 7 (Physical identifier)**.** *The physical identifier of a tuple s is an element of the identifier space $\mathbb{I}$ isomorph to $\mathbb{N}$ and totally ordered. The name of this attribute will be denoted by φ and $s(\varphi)$ designates the value of the physical identifier of s.*

Definition 8 (Tuple-set)**.** *A tuple-set is a countable set of tuples sharing the same schema and including a physical identifier φ. The physical identifier of a tuple is unique in the tuple-set and induces a strict order of the tuples in the tuple-set.*

The common schema of a tuple-set TS is designated by $Attr(TS)$.

Definition 9 (Position of a tuple)**.** *The position of a tuple s in a tuple-set TS is the cardinal of the following set:* $\{s' \in TS / s'(\varphi) < s(\varphi)\}$
It is noted as $\underset{TS}{\text{pos}}(s)$

Data management in sensor environments also requires timestamps to be associated to measured data. We adopt continuous time to allow general data management. This choice is discussed more in section 7.

Definition 10 (Timestamp)**.** *The time-space $\mathcal{T}$ is a field isomorphic to $\mathbb{R}$. The time-space is naturally and totally ordered. A timestamp t is an element of $\mathcal{T}$.*

The notion of batch, presented in [24], is necessary to handle simultaneous tuples (i.e., tuples having the same timestamp). Simultaneous tuples can arise, for example, when joining streams (see definition 32) issued by several sensors[4].

Definition 11 (Batch)**.** *A batch is a tuple-set which contains simultaneous tuples. Given a timestamp, batches form a partition of the set of all tuples having this timestamp. A tuple is part of one single batch, but two simultaneous tuples may belong to two different batches.*
Batch identifiers *are elements of the ordered set $\mathcal{T} \times \mathbb{N}$.*

[4]Clock synchronization is an important issue but is out of the scope of this paper.

3.2. Streams & Relations

As shown in [4, 17] relations and streams are two different concepts that have to be defined separately.

Definition 12 (Temporal Relation)**.** *A temporal relation R is a step function that maps a batch identifier $(t, i) \in \mathcal{T} \times \mathbb{N}$ to a tuple-set $R(t, i)$.*

Example 1: *Buoy deployment* Deployment of a new anchored buoy ($BuoyId = 5$) at time t_1 leads to modifications in relations *Buoys*, *AnchBuoys*, and *Sensors*. Changes in *Buoys* state at time t_1 (batch identifier (t_1,0)) are shown here after:

$Buoys(t_1 - \delta, 0)$:

$BuoyId$	$Drift$
1	0
2	1
3	1
4	0

$Buoys(t_1 + \delta, 0)$:

$BuoyId$	$Drift$
1	0
2	1
3	1
4	0
5	0

In the following, R designates a *temporal relation*. For simplicity, we will use the term *relation* for *temporal relation* (def. 12). The tuple-set $R(t, i)$ is called *instantaneous relation at batch* (t, i) (or *at time* t without any more precision).

Definition 13 (Stream content)**.** *A stream content S is a possibly infinite tuple-set with a schema containing one more special attribute t for a timestamp.*

The physical identifier rules the positional order in streams, and this plays central role in data ordering. Note that the notion of stream is incomplete for now as the stream needs also to be partitioned into batches (see 14,15).

An element of S is a n-tuple s including a value $s(t) \in \mathcal{T}$.

Example 2: Figure 2 illustrates a stream of data sent by sensors located on two buoys. The tuples of the stream have the attributes $Attr(S) = (BuoyID, SensorID, Value, timestamp, \varphi)$.

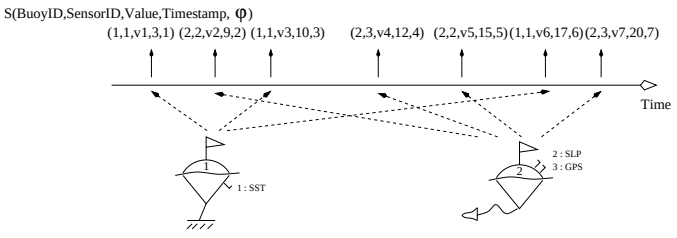


Figure 2: Data from two buoys as a stream. Streams are infinite set of tuples with a common schema.

To define generic operators on streams correctly, a precise definition of tuple order is required. We consider both a temporal order and a positional order. Temporal order is inferred from the timestamps and the positional order from the physical identifiers. As a stream is partitioned into batches, the full definition of a stream needs to consider the function that identifies the batch given a tuple.

Definition 14 (Batch indicator)**.** *A batch indicator is a function that gives the batch identifier of a given tuple taken from a stream content S. $\mathcal{B}_S : S \mapsto \mathcal{T} \times \mathbb{N}$*

Now we can define a stream as a composition of a content and this indicator.

Definition 15 (Stream)**.** *A stream is a couple $(S, \mathcal{B}_S)$ composed of its content and a batch indicator on those tuples.*

In the following, the notation of the stream content S may be used to denote the proper stream $(S, \mathcal{B}_S)$.

The timestamp t_0 will now designates the start timestamp for query evaluation or for temporal relations.

3.3. Batches and positions on streams

We will now assume that, the total order induced by the batches is coherent with the positional order. That is to say: if a tuple s precedes s' in the positional order then $\mathcal{B}_S(s) \leq \mathcal{B}_S(s')$. However, the positional order is strict, e.g. there are no tuples that have the same position on a stream as opposed to the temporal order where two tuples may have the same timestamp. This assumption is not made by [2, 44] but reordered tool/operator [2] is then used to maintain this property. Of course, this hypothesis may be really hard to ensure, especially in low level networks that cannot guarantee order in their messages as in sensor networks. We focus on what is a formal treatment of consistent data streams. Further work could look for the consequences of a violation of this hypothesis.

Hypothesis 1 (Coherency positional order - timestamp order)**.** *The temporal order and the positional order are coherent: Let S be a stream, then*

$$\forall s \in S, s' \in S, \quad s(\varphi) < s'(\varphi) \Rightarrow \mathcal{B}_S(s) \leq \mathcal{B}_S(s')$$

Recall, that tuples have a unique position in a stream. The following definition associates positions and timestamps. Given a tuple position, we can obtain the identifier of the batch containing that tuple.

Definition 16 (Position-timestamp mapping)**.** *Let S be a stream, and $P_S = [\![-1, |S|[\![\subset \mathbb{Z}$. The function $\tau_S : P_S \to \mathcal{T} \times \mathbb{N}$ is the function which given a tuple position returns the identifier of its batch. By convention, $\tau_S(-1) = (t_0, 0)$.*

Corollary 1. *Considering Hypothesis 1, the function: $\tau_S^{-1} : \mathcal{T} \times \mathbb{N} \to \mathbb{Z}$ is the pseudo inverse of τ_S. For a valid batch identifier (t, i) it returns the highest tuple position of that batch. If there is no batch identified by (t, i), the batch*

having the preceding identifier is used.

More formally[5], $\forall (t,i) \in T \times \mathbb{N} \geq (t_0, 0)$,

$$\tau_S^{-1}(t,i) = \begin{cases} |S| - 1 & \text{if} |S| < +\infty \wedge (t,i) \geq \tau_S(|S|-1) \\ \sum_{n=-1}^{|S|-2} n \ \mathbf{1}_{[\tau_S(n), \tau_S(n+1)[}(t,i) & \text{else} \end{cases}$$

As these two functions will be manipulated in the following it is important to state results about their composition.

Property 1 (Properties of τ_S)**.** *The following properties state:*

$$\begin{aligned} \tau_S(0) &\geq (t_0, 0) \\ \tau_S(\tau_S^{-1}(t,i)) &\leq (t,i) \\ \tau_S^{-1}(\tau_S(n)) &\geq n \end{aligned}$$

Moreover, if $\exists s \in S, \mathcal{B}_S(s) = (t,i)$*, then* $\tau_S \circ \tau_S^{-1}(t,i) = (t,i)$.

Proof sketch: The first property states from definition 10 whereas the two others state by the fact that τ_S^{-1} takes the max in case of equality. Results can be found by using the formal definition of τ_S^{-1}. See appendix A for a complete proof of this first proposition.

4. Relational operators

As introduced previously, Astral adopts the two-fold approach of data stream management based on streams and relations. More precisely, this work uses temporal relations (see definition 12) which are fairly different from the classic relations in Codd's algebra [12]. This section discusses how Codds's operators are inherited and proposes adapted definitions of the operators to work with temporal relations. Such operators are necessary to provide clear semantics when handling streams and relations together.

The following sections discuss unary operators, cartesian product, set oriented operators, joins and an operator for domain manipulation which is specific to temporal relations.

4.1. Unary operators

Let us consider selection, projection and renaming defined as *temporal relation* $\rightarrow$ *temporal relation* operators. Their semantics are mainly the usual ones with some small particularities related to temporal relations.

Selection: we note σ_c the selection on temporal relations whereas Σ_c is used for selection on (instantaneous) relations. To take into account the temporal aspect, σ_c has to be defined on batch indicators (t,i). It is defined as follows:

$$\forall R, \ \sigma_c(R) : (t,i) \mapsto \Sigma_c(R(t,i))$$

The evaluation of σ_c at batch indicator (t,i) behaves as the usual selection on the (instantaneous) relation R(t,i).

Projection: For projections a particularity is introduced to preserve the physical identifier (attribute φ) of temporal relations. Were the identifier to be discarded by a projection, φ would be implicitly added: if Π_p with $\varphi \notin p$ is used then $\Pi_{p \cup \{\varphi\}}$ is executed.

Renaming: A constraint on renaming is also introduced to preserve attribute φ: If $\rho_{a/\varphi}$ is used, then the physical identifier is copied to the new attribute a and the original attribute φ remains.

There are no different particularities for the other unary operators. The interested reader may visit Astral's wiki for their definitions.

4.2. Cartesian product

Let us consider now the cartesian product between two temporal relations producing a temporal relation. The main issue in the definition of this operator is the order among tuples and, more precisely, the physical identifier of the resulting tuples.

Recall that the physical identifier allows us to differentiate between tuples having the same values and to order tuples of a tuple-set. As we will see later, this order is important to consider even in relations. Multiple semantics can be used to create the physical identifier of the tuples produced by the cartesian product. To avoid implicit heterogeneity that may impact the final result of a query, we propose to make explicit the creation of the physical identifier in cartesian products. A function $\Phi_{t,i}^{R_1 \times R_2}$ is introduced for this purpose.

The cartesian product for temporal relations is defined as follows.

Definition 17 (Cartesian product)**.** *Let* R_1 *and* R_2*, be two temporal relations such that* $Attr(R_1) \cap Attr(R_2) = \{\varphi\}$,

let (t,i) *be a batch identifier,*

let $\Phi_{t,i}^{R_1 \times R_2} : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ *be an injective function computing the physical identifier of the new tuple given the two physical identifiers of the tuples from* R_1 *and* R_2*.*

The temporal relational cartesian product of R_1 *by* R_2 *at batch* (t,i) *is defined as:* $(R_1 \times R_2)(t,i) =$

$$\bigcup_{\substack{r \in R_1(t,i) \\ s \in R_2(t,i)}} \left\{ \begin{array}{c} r[Attr(R_1) \backslash \varphi] \cup s[Attr(R_2) \backslash \varphi] \cup \\ (\varphi, \Phi_{t,i}^{R_1 \times R_2}(r(\varphi), s(\varphi)) \end{array} \right\}$$

where $r[a_1, ..., a_n]$ *denotes the restriction of tuple* r *to attributes* $a_1, ..., a_n$*.*

[5] As a recall, the function $\mathbf{1}_A$ is the indicator function having the value 1 for all elements of A and the value 0 for all other elements

The difference with respect to the classic cartesian product is the function $\Phi_{t,i}^{R_1 \times R_2}$. This function rules the order (induced by the physical identifier) of tuples in the instantaneous relation $(R_1 \times R_2)(t,i)$ without affecting the composition of the tuples. The choice of function $\Phi_{t,i}^{R_1 \times R_2}$ is nevertheless important because the order's semantic is meaningful for some operators. For example, it happens when the result of the cartesian product is used by a streaming operator (see section 5.1) where the positional order of the tuples in the stream is essential.

The function used in this paper is:

$$\Phi_{t,i}^{R_1 \times R_2}(\varphi_1, \varphi_2) = \varphi_1 * [\max_{r \in R_2(t,i)}(r(\varphi)) + 1] + \varphi_2$$

Although the value of the physical identifier is not relevant by itself, the induced order is important. The preceding function ensures the following property

$$\Phi_{t,i}^{R_1 \times R_2}(a,b) < \Phi_{t,i}^{R_1 \times R_2}(c,d) \Leftrightarrow a < c \vee (a = c \wedge b < d)$$

that describes the strict lexicographic order (which is total) by giving priority to the left side.

There is no direct criteria to argue that the choice made for $\Phi_{t,i}^{R_1 \times R_2}$ is a good one or not. Our choice has been guided by multiple aspects. Firstly, the function provides a total order commonly established over $\mathbb{N}^2$. Secondly, this order reflects the behavior of the usual nested loop algorithm: iterate on R_1 and for each tuple iterate on R_2. Thirdly, due to this simple behavior and formulation, it is commonly found in practice in existing systems.

The characteristics of $\Phi_{t,i}^{R_1 \times R_2}$ may impact the properties of the cartesian product. We highlight the following important fact.

Property 2 (Asymmetrical cartesian product)**.** *The cartesian product is not symmetric in the general case.*

The following example illustrates a cartesian product with the chosen $\Phi_{t,i}^{R_1 \times R_2}$ to calculate the physical identifier. It shows the asymmetry of the cartesian product.

Example 3: Consider relations R_1 and R_2, both containing a sensor *id* and their sensed values – temperature *tv* or humidity *hv*.

$R_1(t,i)$

φ	id	tv
0	1	20
1	3	23
2	42	22

$R_2(t,i)$

φ	id2	hv
0	42	45
1	2	50

$R_1 \times R_2$ and $R_2 \times R_1$ differ as shown here after.

$(R_1 \times R_2)(t,i)$

φ	id	id2	tv	hv
0	1	42	20	45
1	1	2	20	50
2	3	42	23	45
3	3	2	23	50
4	42	42	22	45
5	42	2	22	50

$(R_2 \times R_1)(t,i)$

φ	id	id2	tv	hv
0	1	42	20	45
1	3	42	23	45
2	42	42	22	45
3	1	2	20	50
4	3	2	23	50
5	42	2	22	50

In many systems and formalizations, the aspects discussed in this section have been ignored or suggested only. We highlight such kind of ambiguity, find properties and point out potential problems in order to propose a precise formal framework.

4.3. Sets operators

This section discusses the union and difference of temporal relations.

Definition 18 (Set difference)**.** *Let R_1 and R_2 be two temporal relations with the same schema. The set difference is defined as:*

$$R_1 - R_2 : t, i \mapsto R_1(t,i) - R_2(t,i)$$

There is no particularity in the difference of temporal relations. Nevertheless, it is worth noting that the physical identifier (attribute φ) is handled as any other attribute. This implies that two tuples are considered as being equal when their attribute values are the same even for attribute φ.

The definition of the union for temporal relations is a little more complicated because the resulting tuples require a meaningful physical identifier. As seen for the cartesian product, a problem of semantics arises in the resulting order.

Definition 19 (Set union)**.** *Let R_1 and R_2 be two temporal relations with the same schema.*

Let $\Phi_{t,i}^{R_1 \cup R_2} : \{0,1\} \times \mathbb{N} \to \mathbb{N}$ be a function that will ensure a coherent order of φ at each instant (t,i) i.e.

The set union is defined as:

$R_1 \cup R_2 : t, i \mapsto$

$$\cup_{r \in R_1(t,i)} \left\{ r[Attr(R_1) \backslash \varphi] \cup (\varphi, \Phi_{t,i}^{R_1 \cup R_2}(0, r(\varphi)) \right\}$$
$$\cup_{s \in R_2(t,i)} \left\{ s[Attr(R_2) \backslash \varphi] \cup (\varphi, \Phi_{t,i}^{R_1 \cup R_2}(1, s(\varphi)) \right\}$$

Several semantics can be considered for $\Phi_{t,i}^{R_1 \cup R_2}$. Note that using

$$\Phi_{t,i}^{R_1 \cup R_2}(i, \varphi) = \varphi$$

would lead to a symmetric union, but without unicity of φ values in the result. This may happen for tuples of R_1 and R_2 having the same value for φ but no exact match for all the other attributes. Such a function can not be used because a total order inferred by φ is required.

For a general case, we will consider the following function

$$\Phi_{t,i}^{R_1 \cup R_2}(j, \varphi) = j * [\max_{r \in R_1(t,i)}(r(\varphi)) + 1] + \varphi = \Phi_{t,i}^{R_2 \times R_1}(j, \varphi)$$

Here, as for the cartesian product, the value of φ itself is not relevant but the induced order is important. The chosen function gives priority to the left hand side relation (their tuples are placed first). The symmetric property does not hold for the union.

Such ordering problems have not yet been clarified by the literature and are important because they impact the result of queries involving streams.

4.4. The joins

Given the preceding definitions, particularly the selection and cartesian product of temporal relations, the formalization of joins is now straightforward. The correspondence between joins and cartesian products for temporal relations is the same as in relational algebra.

Definition 20 (Natural join)**.** *Let R_1 and R_2 be two temporal relations. Let $\{a_1, ..., a_n\}$ be their common attributes except φ. Let $\{b_1, ..., b_n\}$ be a list of n temporary attributes neither defined in R_1 nor in R_2.*

The natural join is defined as $R_1 \bowtie R_2 =$

$$\Pi_{Attr(R_1)\cup Attr(R_2)}(\circ_{i=1}^{n}\sigma_{a_i=b_i})\left(R_1 \times (\circ_{i=1}^{n}\rho_{b_i/a_i})(R_2)\right)$$

Note that this definition (using selection, renaming and cartesian product) is exactly the same as in classical relational algebra. It would also be similar for θ-join ($R_1 \bowtie_\theta R_2$) and semi-join ($R_1 \ltimes R_2$). Using the set operations defined in the previous section, outer joins, antijoin and division can also be expressed similarly.

4.5. Using the past states of a temporal relation

The preceding sections focused on the usual relational operators applied to the temporal relations introduced in Astral. This section introduces a new operator, specific to temporal relations which are considered as functions.

Let O be one of the already introduced operators. For all (t, i), the result of $O(R_1, ..., R_n)(t, i)$ is computed with $R_1(t, i)$, ..., $R_n(t, i)$. One important feature is to be able to use, not only the states of relations at time (t, i) but also at the past states ($(t', i') < (t, i)$) to compute results. To do this, a new operator is required. It allows us to change the domain of the R function so as to use batch indicators of the past and to refer to the past states of relations.

Definition 21 (Domain manipulator)**.** *Let R be a temporal relation, let $f : \mathcal{T} \times \mathbb{N} \to \mathcal{T} \times \mathbb{N}$ be a function of time indicators (time transformer), and c a condition over $\mathcal{T} \times \mathbb{N}$.*

If, $\forall (t, i) \in \mathcal{T} \times \mathbb{N}, c(t, i) \wedge f(t, i) \leq (t, i)$,

Then, the domain manipulator is defined as follows:

$$D_c^f(R) = t, i \mapsto \begin{cases} R(f(t,i)) & \text{if } c(t,i) \\ \emptyset & \text{else} \end{cases}$$

This operator, not yet considered by the existing proposals, is interesting as it allows the creation of a state based on one or several past states as a "current" state of a relation. This can be used, for example, to compare a relation with its past states. The first direct application may be the following one.

Definition 22 (Fixed relation)**.** *Let R be a temporal relation and let t_s be a timestamp. The fixed relation R at t_s is defined as*

$$R^{t_s} = D_{t\geq t_s}^{t,i\mapsto t_s,0}(R) = t, i \mapsto \begin{cases} R(t_s, 0) & \text{if } t_s \leq t \\ \emptyset & \text{else} \end{cases}$$

Here R^{t_s} is the relation R *frozen* at time t_s. The following states of R can be compared to this past state. The *fix relation* operator is useful to express continuous queries using instantaneous queries for designation (see definition 5 and section 7.1). That is, an instantaneous query is used to select and "fix" the set of sensors to be used in the evaluation of the continuous query. Another application of such an operator will be seen when manipulating streams with joins (see definition 33).

We have now explored the core relation-to-relation operators. We have seen that such formalization work allowed us to reveal existing semantic discrepancies in data management which were hardly perceivable but which were nevertheless significant. With our model, we can target semantic problems and be aware of them while using the operators.

5. Stream operators

This section presents the operators to manipulate streams. It first presents the *streamers* (see Section 5.1) which create streams from relations. Windows operators, allowing the creation of relations from streams, are defined in Section 5.2. These two types of operators (streamers and windows) are the basis for defining the join operators of streams (see Section 5.3)[6].

5.1. Streamers

Streamers are operators used to create a stream from a relation. Streams can be generated by monitoring relations. Two types of streamers can be distinguished: those fed when changes occur in the relation (called *sensible* streamers) and those that do not "react" to changes in the relation (called *independent* streamers).

Streamers are in charge of *stamping* tuples of the streams. This stamping is both positional and temporal and so must ensure that coherent orders are produced. We define the stamping function as the function that adds a timestamp and a position to a tuple of a relation.

Definition 23 (Streamer stamping)**.** *Let R be a relation, let (t, i) be a batch identifier, let $\Phi_{t,i}^S : \mathbb{N} \to \mathbb{N}$ be a strictly increasing function that ensures:*

$$\forall (t', i') \leq (t, i), \Phi_{t',i'}^S \leq \Phi_{t,i}^S.$$

The function that stamps tuples from $R(t, i)$ is defined by: $\forall s \in R(t, i), \forall t_s \in T,\ \Psi_{t,i}(s, t_s) =$

$$\{('t', t_s), (\varphi, \Phi_{t,i}^S(\varphi))\} \bigcup_{a\in Attr(R)\setminus\{\varphi,'t'\}} \{(a, s(a))\}$$

Such a definition ensures that the stamped tuples that will form a stream verify the following properties:

[6] For space reasons, some technical details are not included in the paper but can be found on the Astral's Wiki.

- each tuple has a timestamp t_s, which is the moment when it has been stamped,
- each tuple has a physical identifier (position) greater than the identifiers of the previous stamped tuples,
- the positional order of $R(t,i)$ is preserved in the final stream.

If a timestamp was defined in the original tuples, it will be replaced in the stamping process. It is possible to conserve the value of the original timestamp by using an operator $\rho_{t_{original}/t}$ before the streamer that will stamp the tuples.

The actual expression of $\Phi^S_{t,i}$ is not important. Its purpose is to insure a coherent positional order derived from the physical identifier of the instantaneous relation. Stamping using the creation time is crucial. In sensor applications, the question of which timestamp has to be used for tuples containing aggregated measures (5min range for instance) has been discussed many times [18, 7, 16, 20]. Three usual choices have been identified: *min* or *max* of the timestamps of the involved tuples and *user-defined*. Several strong arguments can be given for choosing the moment of creation[7] of the tuple as its timestamp. Firstly, this choice is coherent with the time when the value has been calculated. Secondly, the order is preserved whereas order violations could be introduced by taking another value. Thirdly, this choice does not make any hypothesis on the schema of R and allows time-stamping of tuples without timestamp originally. Property 3 in Section 6 will show how an original timestamp can be transmitted to a final stream.

Sensible streamers can now be defined. Given a relation, sensible streamers adds to a stream changes occurring in the relation.

Definition 24 (Sensible streamers)**.** *Let R be a relation, Let (t,i) and $(t,i)^-$ be such that $(t,i)^- < (t,i)$. $(t,i)^-$ is a batch identifier infinitely near (but not equal to) (t,i).*[8]
The core sensible streamers are:

- *$I_S(R)$ the operator giving the stream of tuples inserted in the relation R:*
  $I_S(R) = S$: $s' = \Psi_{t,i}(s,t) \in S,\ \mathcal{B}_S(s') = (t,i) \Leftrightarrow s \in R(t,i) \wedge s \notin R((t,i)^-)$
- *$D_S(R)$ the operator giving the stream of tuples suppressed from the relation R:*
  $D_S(R) = S$: $s' = \Psi_{(t,i)^-}(s,t) \in S,\ \mathcal{B}_S(s') = (t,i) \Leftrightarrow s \notin R(t,i) \wedge s \in R((t,i)^-)$
- *$R^u_S(R)$ the operator giving the stream of the content of the relation R every time it changes:*
  $R^u_S(R) = S$: $s' = \Psi_{t,i}(s,t) \in S,\ \mathcal{B}_S(s') = (t,i) \Leftrightarrow R(t,i) \neq R((t,i)^-) \wedge s \in R(t,i)$

[7] Greater or equal to the *max*

[8] Choosing such timestamp is always possible as R is a "step function"

Example 4: Considering the buoys example, relations Buoys and AnchBuoys are as illustrated by figure 1 (in Section 2.1). Attribute BuoyId may be used for a natural join. Let

$$R = Buoys \bowtie AnchBuoys$$

be the relation which gives meta-data for anchored buoys. The deployment or pickup of a buoy at time t leads to an update in R at this time.

- The stream of buoy arrivals in the system (when new buoys are deployed) can be express as: $I_S(\Pi_{BuoyId}R)$.
- The stream of departure (buoy pickup or termination of observation) is given by: $D_S(\Pi_{BuoyId}R)$.
- The stream of localization of anchored buoy 42:

$$R^u_S(\Pi_{loc}\sigma_{BuoyId=42}(R))$$

Let introduce the relation M containing the data send by each buoy: $l \in M \Leftrightarrow \{l = (id, m, \varphi)\ \}$ where id is the sensor id, m the received measurement and φ the physical identifier. Receiving data from a buoy at time t leads to an insert in M at this time, the φ being incremented at each insertion. Figure 3 shows M at insertions times $(3, 9, 10, 12, ...)$. In this example, the $\Phi^S_{t,i}$ function (cf. 23) is define as follows :

$$\Phi^S_{t,i}(\varphi) = \varphi$$

The stream of sensed data is given by $I_S(M)$.

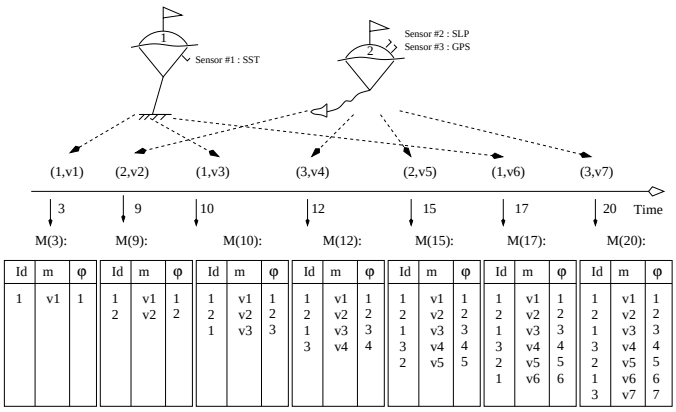


Figure 3: Stream from two buoys can be interpreted as created from the operator I_S.

The rate of tuples in sensible steamers is dictated by the updates on the temporal relation. Let's now introduce *independent streamers* which create tuples at their own rate.

Definition 25 (Independent streamer R^r_S)**.** *Let R be a relation, and r be a period of time.*

The independent streamer that sends the content of R each r period is defined as: $R^r_S(R) = S$:

$$\Psi_{t,i}(s,t) \in S,\ \mathcal{B}_S(s) = (t,i) \Leftrightarrow s \in R(t,i) \wedge t - t_0 \in r\mathbb{N}$$

Example 5: Continuing example 4 and using relation Sensors illustrated by figure 1 (in Section 2.1), the stream of available measurements every second can be written as: $R^{1s}_S(\Pi_{m,Id,BuoyId}(M \bowtie Sensors))$. The rate of tuples in the stream is 1 second independently of changes in the relations.

Other independent streamer operators could be defined to create streams of updates/deletes/inserts every period r. This can be done by comparing the state of the relation at t and $t - r$. Similarly, *insert*-sensitive streamers that send the content of a relation at each insertion, could also be defined. We don't introduce them in this paper because a more complete study of their utility in practice still necessary.

Streamers have already been defined in other algebra like in STREAM [4]. For instance, I_S is similar to $ISTREAM$ and D_S is similar to $DSTREAM$. $RSTREAM$ is similar to R^δ_S, with δ being the chronon of the system (1s for instance). As $RSTREAM$ uses implicitly the chronon of the system, comparing results given by two systems using two different chronons (for example δ_1 and δ_2) may lead to some confusions. The exact behavior can be expressed in Astral with $R^{\delta_1}_S$ and $R^{\delta_2}_S$.

5.2. Windows

Window operators can be used to create relations from streams. These relations can then be used to create a stream of aggregate values (min, max,...) and also to define join between streams. A window may be temporal (e.g. data from the last 10 minutes), or positional (e.g. the 10th last data of a stream) or cross domain (e.g. the 10th last data every 10 minutes).

In literature [9] mains windows are:

- *Fixed* : boundaries are fixed, the window is evaluated only once.
- *Sliding* : width is fixed and boundary linearly moving
- *Tumbling* : is a particular sliding window, the boundaries move is equal to the width of the window so that intersection between windows is empty.
- *Landmark* : lower boundary is fixed and the upper one grows linearly.

We give here a more general definition of windows, more detail can be found in [35]. The first step is to define a sequence of windows and the operator to create it.

In general, a fixed window is defined by a lower and an upper bound. Such bounds delimit the subset of the data stream *observed* in the window. In data stream management, sequences of windows are required to observe the data in the evolving stream. This section defines the operator to create sequences of windows (definition 28). It is based on a description of such a sequence (definition 26) and its correlation to data streams (definition 27). This correlation uses batches (introduced in section 3.1) that support positional and temporal windows in the presence of simultaneous tuples in the stream.

Definition 26 (Window Sequence Description)**.** *Let $\mathcal{D}$ and $\mathcal{D}'$ be either $\mathcal{T}$ or $\mathbb{N}$, a Window Sequence Description is a triplet (α, β, r) where:*

- *$r \in \mathcal{D}$ is the boundaries evaluation rate*
- *α and β are two functions from $\mathbb{N} \to \mathcal{D}'$ representing the boundaries evolution.*

$\alpha(j)$ and $\beta(j)$ define the j^{th} values for the boundaries. The first values are given for $j = 0$.

These functions must verify the following properties (stated here for $\mathcal{D} = \mathcal{D}' = \mathcal{T}$):

$$\forall j \in \mathbb{N}, \begin{cases} \alpha(j) \leq \beta(j) & \textit{beginning before end} \\ \alpha(j) \geq t_0 & \textit{beginning exists} \\ \beta(j) \leq jr + \beta(0) & \textit{end accessible} \end{cases}$$

By applying τ_S (or τ_S^{-1}), these conditions for other values of $\mathcal{D}$ and/or $\mathcal{D}'$ are easy to find.

Example 6: Let's consider the stream of the SST sensor 42. In order to monitor temperature, we consider that per each 100 passed items, we have to extract the last 10 items. For this case, we require a sequence of positional windows generated for every 100 items ($r = 100$) where each window contains the 10 last items. We handle full positional windows, $\alpha, \beta \in (\mathbb{N} \to \mathbb{N})^2$. The first window covers from the 91th item to 100th item: $\alpha(0) = 91$ and $\beta(0) = 100$. The boundaries evolve linearly as follows:

$$\begin{aligned} \alpha(j) &= 100j + 91 \\ \beta(j) &= 100(j+1) \\ r &= 100 \end{aligned}$$

Window creation requires mapping of the stream tuples into the corresponding window based on the given sequence description. For this, we use the following delaying function which includes batch identifiers. Based on the window sequence description, this function gives the "rank" of the last created window at the moment indicated by the given batch identifier.

Definition 27 (Delaying function)**.** *The delaying function is a function from $\mathcal{T} \times \mathbb{N} \to \mathbb{Z}$ that maps the batch identifier with the id of last valid boundaries.*

- *If $r \in \mathcal{T}$, this function is $\gamma : t, i \mapsto \left\lfloor \frac{t - \beta(0)}{r} \right\rfloor$.*
- *If $r \in \mathbb{N}$, this function is $\gamma : t, i \mapsto \left\lfloor \frac{\tau_S^{-1}(t,i) - \beta(0)}{r} \right\rfloor$.*

The term delaying *is related to the fact that the j^{th} boundaries computation has to be delayed until $\gamma(t,i) = j$.*

Example 7: In the last example, the delaying function is $\gamma(t,i) = \left\lfloor \frac{\tau_S^{-1}(t,i)}{100} \right\rfloor - 1$. If we consider that the SST sensor 42 emits a tuple per second. As $\gamma(1024,0) = \lfloor \frac{1024}{100} \rfloor - 1 = 9$. The 10th window is the last created window at this batch identifier.

Given this, it is now possible to define an operator that generates a temporal relation from a stream. This temporal relation has a sequence of states. Transitions between states are triggered by batches arrivals and changes of γ.

Definition 28 (Window Sequence Operator)**.** *Let S be a stream, (α, β, r) be a Window Sequence Description and γ be the delaying function associated to this description,*

The Window Sequence Operator is defined as :

- $\forall (t,i)$, *such that* $\gamma(t,i) \geq 0$,

  If the description has temporal bounds:

$$S[\alpha,\beta,r](t,i) = \{s \in S/\ (\alpha(\gamma(t,i)),0) \leq \mathcal{B}_S(s) \leq (\beta(\gamma(t,i)),i)\}$$

  If the description has positional bounds:

$$E_{t,i} = \{s \in S/\ \tau_S(\alpha(\gamma(t,i))) \leq \mathcal{B}_S(s) \leq \tau_S(\beta(\gamma(t,i)))\}$$

$$S[\alpha,\beta,r](t,i) = \{s \in E_{t,i}/ \ |\{s' \in E_{t,i}/ s(\varphi) < s'(\varphi)\}| \leq \beta(\gamma(t,i)) - \alpha(\gamma(t,i))\}$$

- $\forall (t,i)$, *such that* $\gamma(t,i) < 0$, $S[\alpha,\beta,r](t,i) = \emptyset$

This definition distinguishes the case of temporal windows from positional windows.
For temporal windows, it considers the batches ranging:

- **from** the first batch providing tuples falling in the window scope (noted $(\alpha(\gamma(t,i)),0)$), i.e. their timestamp is $\geq$ than the lower bound of the window
- **until** the last batch providing tuples in the window scope (noted $(\beta(\gamma(t,i)),i)$). This is the ith batch having the maximum timestamp inferior than the higher bound of the window.

Note that the relation $S[\alpha,\beta,r]$ may change at batch arrival even if time doesn't change. Using the batch identifier to define a window is mandatory since the partitioning introduced by batches is needed when building a new stream from window. Disregarding batch identifiers would lead to the loss of batch grouping. A treatment exclusively based on timestamps would not be sufficient.

For positional windows, the case where batches contain exactly one tuple each (full spread stream) is simpler than the general case where batches contain more tuples.

- For full spread streams, $E_{t,i} = S[\alpha,\beta,r](t,i)$.
- For the general case, $E_{t,i}$ is composed of all batches including tuples falling in the current window scope. The instantaneous relation $S[\alpha,\beta,r](t,i)$ is a subset of $E_{t,i}$.

  As example, consider "1-tuple" width windows. $E_{t,i}$ contains the last batch, but as it may include several tuples, only one has to be selected. The selection over the more-recent tuples is done with regard to the positional order φ.

- The γ function in the positional case is driven by τ_S^{-1} which provides the maximum tuple position in case of conflict. This choice is important as other choices would introduce tuple loosing on simultaneous events.

In the following, to specify the rate r we will use n for positional or a temporal unit s, ms, m.

Example 8: Sliding Windows: Figure 4 shows a sliding window. It slides of two seconds every two seconds with a constant width of 3 seconds. $t_0 = 0$ for simplicity.

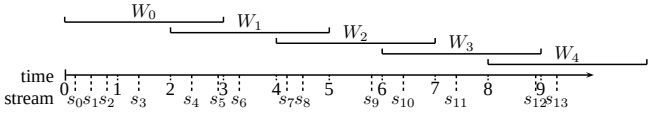


Figure 4: Sequence of sliding windows «PERIOD 3s SLIDE 2s»

We have $r = 2s$ as sliding rate. For the first window : $\alpha(0) = 0$ and $\beta(0) = 3$. The slides of boundary is 2s so:

$$\forall i \in \mathbb{N},\ \begin{cases} \alpha(i) & = i * 2s + t_0 \\ \beta(i) & = i * 2s + 3s + t_0 \end{cases}$$

The temporal relation generated from the stream S can be noted : $S[2is, 2is + 3s, 2s]$.

Given a timestamp $t = 5.5$ it is easy to compute $S[\alpha,\beta,r](5.5)$. The windows that can be computed at this time is the one numbered $\gamma(5.5) = \left\lfloor \frac{5,5-\beta(0)}{r} \right\rfloor = 1$. So $S[\alpha,\beta,r](5.5) = W_1 = \{s_4, s_5, s_6, s_7, s_8\}$

Definition 29 (Partitioned window)**.** *A sequence of partitioned windows is build as the union of windows sequences on a stream partitioned by a set of attributes a_1, ... , a_k:*

$$S[a_1 ... a_k/\alpha,\beta,r] = \bigcup_{i \in Dom(a_1,...,a_k)} (\sigma_{(a_1,...,a_k)=i} S)[\alpha,\beta,r]$$

This operator is useful to create the same sequence of windows on sub-streams of a main stream. See example bellow.

Example 9: Given the stream S' of measurements defined in example 4, the last value sended by each buoy is given by $S[BuoyId/i, i, 1n]$ whereas the last value sended by each sensor is given by $S[id/i, i, 1n]$.

The proposed model is generic enough and can be used to create complex windows like the ones generated by the used of more algorithmic models [10]. More definitions

	Definition	Equivalence
$[B]$	$[\tau_S^{-1}(\tau_S(i)^-)+1, i, 1n]$	$\{s \in S, \mathcal{B}_S(s) = \tau_S \circ \tau_S^{-1}(t,i)\}$
	The slide-by-batch sequence of windows produces the last seen batch in the stream	
$[L]$	$[i, i, 1n]$	
	The slide-by-tuple sequence of windows, where each window contains only the last n-tuple. This sequence is equals to $[B]$ when the stream has a one tuple per batch policy.	
$[\infty]$	$[0, i, 1n]$	$\{s \in S, \mathcal{B}_S(s) \leq (t,i)\}$
	The sequence of cumulative windows containing all the stream up to the current batch	

Table 1: List of notable windows

about aperiodic windows have been presented in a preliminary work [35].

The table 1 presents notation adopted for a set of very useful windows. The table describes their meaning and also provides an algebraic equivalent of the content at (t,i)[9].

5.3. Relational operators on streams

This section considers relational operators on streams. It presents general definitions of the operators which cover existing proposals (see Section 7 for discussions on related work).

5.3.1. Unary operators on streams

The three basic unary relational operators selection, projection and renaming are defined for streams. The definition of the selection is as follows.

Definition 30 (Selection on stream)**.** *Let S be a stream, the selection of tuples in S is defined as:*

$$s \in \sigma_c S \Leftrightarrow s \in S \wedge c(s)$$
$$\textit{and}$$
$$\forall s \in \sigma_c S, \mathcal{B}_{\sigma_c S}(s) = \mathcal{B}_S(s)$$

The particularity of the definition of the operators for streams concerns the batch identifier. The choice we made is to leave the batch identifier unchanged in the result. The same applies to projection and renaming without any difficulty.

It is worth noting that the unary operators can also be expressed using windows, streamers and relational operators (see corollary 3 in Section 6). This shows that stream-to-stream operators can be seen as a composition of other operators.

[9]The equivalences are non-trivial properties

5.3.2. Cartesian product and joins

The definition of the cartesian product and joins involving a stream are build using windows and relational operators to obtain a temporal relation. A streamer is then used to build the resulting stream from this temporal relation.

Definition 31 (Cartesian product stream×relation)**.** *Let S be a stream and R be a relation, the cartesian product operator (stream×relation)→stream generates as output stream, the result of a product between the relation and the last tuples of the stream:*

$$S \times R = I_S(S[B] \times R)$$

The main choices in this definition concern the windows to be used and the streamer:

- The preceding definition uses window $[B]$ which designates the last batch of the stream. The last tuples of the stream are used in the product with the relation. This is a natural interpretation.
- The preceding definition uses streamer I_S which put in the stream the new inserted tuples. This leads to a consistent stream with no duplicates. For instance, when only R is updated (new tuples are inserted) then only the newly created tuples will be sent. The old join-tuples will not be sent again in the stream. The use of I_S is also interesting because of the associativity property (see section 7).

Analogous to cartesian product, let's define the join operator as follows.

Definition 32 (Join stream×relation)**.** *Let S be a stream and R be a relation, the join operator (stream×relation)→stream generates as output stream, the result of a join between the relation and the last tuples of the stream:*

$$S \bowtie R = I_S(S[B] \bowtie R)$$

The choice concerning the window and streamer used in the join definition are the same than in the cartesian product for the aforementioned reasons. Note that any kind of window W and streamer Sc could have been used for both the cartesian product and the join. Definition 32 can be seen as an instance of a generic definition:

$$S \bowtie_c R = Sc(S[W] \bowtie R)$$

In definition 32, the result stream contains tuples triggered by updates in R and tuple arrival in S. Let's now consider a variant where new join tuples feed the output stream only when new tuple arrives in S but not when R is updated.

Definition 33 (Semi-sensitive join stream×relation)**.** *The semi-sensitive-join operator (stream×relation)→stream generates as output stream the*

result of a join between the last tuples of the stream and the relation at the time equal to the last batch of the stream:

$$S \ltimes R = I_S(S[B] \bowtie D^{\tau_S \circ \tau_S^{-1}}_{t \geq t_0}(R))$$

This definition involves four core operators: streamer, window, join and domain manipulator.

Example 10: Considering the example given in figure 1, we want to produce the stream that has the following schema $(id, BuoyId, Type, Unit, value, t)$, in which, a new tuple appears each time a new tuple appears in the stream of data (S). This behaviour is exactly equals to:

$$S \ltimes Sensors$$

Using an usual stream-to-relation join, the streamer will also produce a new tuple each time *Sensors* is updated (new sensor has arrived, unit has changed,...) leading thus to redundant values in the resulting stream.

Now, the join operation (stream×stream)→stream is defined using the previous definitions.

Definition 34 (Band join stream×stream)**.** *Let S_1 and S_2 be two streams and d be a time interval, the join operator (stream×stream)→stream is defined as the join between cumulative windows on each stream for n-tuples with a timestamp differing of at most* d.

Let's note $\Pi^{\bowtie}(R) = \Pi_{Attr(R)\setminus\{t'\}}(R)$

$$\begin{aligned} S_1 \overset{d}{\bowtie} S_2 &= I_S(\Pi^{\bowtie}(S_1[\infty] \underset{|t'-t| \leq d}{\bowtie} (\rho_{t'/t} S_2[\infty]))) \\ &= \{\Psi_{t,i}(s_1 \times s_2, \max(s_1(t), s_2(t))) \\ &\qquad s_1 \in S_1, s_2 \in S_2, \\ &\qquad |s_1(t) - s_2(t)| \leq d\} \end{aligned}$$

As mentioned for the preceding operators, other choices of window creation and streamer can be made for joins. Definition 34 has been adopted because it is quite intuitive and covers most used application needs. Highlighting the condition d allows simple notation for mostly all existing joins in streaming literature and can be used to tolerate timestamp divergence. It is worth noting that, instead of using a test on timestamps, a test on the positions can be used.

To illustrate the expressiveness of the proposed formulation, let's show how it can be used to provide two particular join operators defined in the literature.

- The instantaneous join proposed in [23] can be expressed as follows:

$$\begin{aligned} S_1 \overset{0}{\bowtie} S_2 &= I_S(S_1[\infty] \bowtie S_2[\infty]) \\ &= I_S(S_1[B] \bowtie S_2[B]) \end{aligned}$$

  This join generates an n-tuple in the output stream when equal timestamps are found in cumulative windows.

- The infinite join proposed in [43] can be expressed as follows:

$$S_1 \overset{\infty}{\bowtie} S_2 = I_S(\Pi^{\bowtie}(S_1[\infty] \bowtie \rho_{t'/t} S_2[\infty]))$$

  This join generates as output stream the relational join of the cumulative windows.

5.4. Spread

Finally, we will investigate the spread operator, the only core stream-to-stream operator of Astral. This operator was originally introduced in the famous paper about stream processing standardization [24]. It manipulates the ordering of simultaneous tuples inside a stream by redefining the batches.

The following definition allows us to spread, or refine, the different batches using the criteria over some attributes.

Definition 35 (Spread operator)**.** *Let S be a stream, let $a_1, ..., a_n$ be attributes included in $Attr(S)$, then the stream S spread by $a_1, ..., a_n$ noted $\rhd_{a_1,...,a_n} S$ verifies:*

$\forall s_1, s_2 \in S^2$, then $s_1, s_2 \in (\rhd_{a_1,...,a_n} S)^2$ and $\mathcal{B}_{\rhd_{a_1,...,a_n} S}(s_1) < \mathcal{B}_{\rhd_{a_1,...,a_n} S}(s_2)$

$$\Leftrightarrow \begin{cases} \mathcal{B}_S(s_1) < \mathcal{B}_S(s_2) \\ or \\ \mathcal{B}_S(s_1) = \mathcal{B}_S(s_2) \wedge s_1[a_1, ..., a_n] < s_2[a_1, ..., a_n] \end{cases}$$

Tuple comparison uses the lexicographical order on the given list of attributes.

Here we can see that this operator does not changes the content of the stream and only affects the batch indicator.

Example 11: Suppose that in the stream of data S of our running example, we have the following content: $S = \{s_1 = (1, 42, t_0), s_2 = (2, 43, t_0), s_3 = (3, 75, t_0)\}$ and those tuples are inside the batch $(t_0, 0)$.

If we want to have a window that has one tuple per state, using directly $(S, \mathcal{B}_S)$ it is not possible. A description $S[B]$ (last batch) would only produce one state $(t_0, 0)$ with all the tuples. A description $S[L]$ (last tuple), would produce only $S[L](t_0, 0) = \{s_3\}$ as it has to choose one tuple over the three inside the batch (we assume here that s_3 has the highest rank).

Let S' be the stream defined as: $S' = \rhd_{id} S$. Consequently, the batch $(t_0, 0)$ of S will be spread considering the different values of id. The content of S' stays the same but we will have: $\mathcal{B}_{S'}(s_1) = (t_0, 0)$, $\mathcal{B}_{S'}(s_2) = (t_0, 1)$ and $\mathcal{B}_{S'}(s_3) = (t_0, 2)$. Hence, using the window $S'[B]$, we will have three consecutive states $(t_0, 0)$, $(t_0, 1)$ and $(t_0, 2)$ having respectively the tuples $\{s_1\}$, $\{s_2\}$ and $\{s_3\}$.

Similarly, the spread all $(\lhd_{a_1,...,a_n} S)$ can be defined. The basis of this operator is the same but instead of splitting the current batch in multiple ones, it regroups all the batches that has the same timestamp and then apply the spread.

In the specifications [24] the spread operation can also be submitted without any attributes. This special case specifies that any strict total order is acceptable as long as it respects the source batches (or the timestamp in the spread all case). In Astral, in order to keep a coherence,

this case will be the same as $\rhd_\varphi$ which will spread by the rank, therefore there will be one and only tuple per batch. We could have defined an ordering function ruled by each implementation, but it seemed easier and far more comprehensible that way. In order to simplify the notations, $\rhd$ and $\lhd$ are acceptable for this last case.

Notation table

Table 2 presents a summary of the notation used in Astral.

Notation	Quick Description
t	Timestamp attribute
φ	Physical attribute
s, $s(a)$	Tuple and its value for a
$s[a_1, ..., a_n]$	Tuple restriction
$Attr(TS)$	Common schema of a tuple-set
$\underset{TS}{\text{pos}}(s)$	Position of a tuple in a tuple-set
$(t, i) \in \mathcal{T} \times \mathbb{N}$	Batch identifier
t_0	Starting timestamp
R	(Temporal) Relation
$R(t, i)$	Instantaneous Relation
S	Stream/Stream Content
$\mathcal{B}_S$	Batch indicator
τ_S, τ_S^{-1}	Position-timestamp mapping
$\Pi_a(R)$	Projection to the attributes list a
$\sigma_c(R)$	Selection with the condition c
$\rho_{b/a}(R)$	Rename attribute a to b
$R_1 \times R_2$	Relational cartesian product
$R_1 \bowtie R_2$	Relational natural join
$R_1 \cup R_2$ ($\cap$, $-$)	Union (disjonction, difference)
$D_c^f(R)$	Domain manipulation
R^{t_f}	Time fixation of R at t_f
$I_S(S)$, $R_S^u(S)$	Sensible streamer
$R_S^r(S)$	Independant streamer with rate r
$S[\alpha, \beta, r]$	Window operation
$S[a/\alpha, \beta, r]$	Window partitioned by a
$[L]$, $[B]$, $[\infty]$	Special descriptions (see table 1)
$S \bowtie R$	Natural stream-relation join
$S \ltimes R$	Semi-sensitive join
$S_1 \bowtie^d S_2$	Stream band join width d
$\lhd_a S$, $\rhd_a S$	Spread (all) operator
$\Phi_{t,i}^{R_1 \mathcal{O} R_2}$	Order function of the operation $\mathcal{O}$
$\Psi_{t,i}$	Stamping function for streamers
γ	Delaying function for windows
$E_{t,i}$	Scope of a window at t, i

Table 2: Notation Table

6. General properties

This section presents the main properties of the algebra. They are particularly useful in query rewriting and optimization. Section 6.1 presents important results on *timestamp transmission*. It shows which windows and streamers operate without any effect on original timestamps. Section 6.2 presents results concerning the associativity of operators. This is very important as, for example, it allows us to push selections and projections down in the query tree. It is shown here when associativity holds and when it doesn't[10].

Operator sharing between queries is also important for optimization purpose. The *transposition* property introduced in Section 6.3 can be used to identify whether windows can be shared by two queries. The complete proof for all the propositions are available in appendix A.

In the following two tuple-sets are considered equivalent if they contain the same tuples (except for the physical identifier) in the same relative order.

Definition 36 (Tuple-set equivalence)**.** *Let TS_1 and TS_2 be two tuple-sets. TS_1 and TS_2 are equivalent ($\equiv$) if and only if their tuples compared in order two by two are identical except for φ.*

6.1. Properties on windows and streamers.

Definition 23 shows that tuples produced by a streamer are time-stamped and the original timestamp may be lost. The property presented hereafter, identifies cases when the streamer stamping does not affect the original timestamp. Here, the original timestamp is *transmitted* through the windows and the streamer operators.

Property 3 (Timestamp transmission for I_S and R_S^u)**.** *Let S be a stream and s be a tuple of S, $[\alpha, i+k, 1n]$ a window sequence description (WSD) with α increasing.*

Considering $S' = I_S(S[\alpha, i+k, 1n])$: if a stamped tuple of S' is in the batch (t, i) then, this tuple were originally in the batch (t, i) of S.

That is:

$$\Psi_{t,i}(s,t) \in S' \wedge \mathcal{B}_{S'}(\Psi_{t,i}(s,t)) = (t,i) \Rightarrow \mathcal{B}_S(s) = (t,i)$$

This property also stands for $R_S^u(S[B])$.

As for I_S the timestamp transmission applies for WSD of form $[\alpha, i+k, 1n]$, the property states for sequences $[B]$ and $[\infty]$ which are particular cases of the general form.

Considering property 3, the following equivalences with the original stream S derive.

Corollary 2 (Equivalence of streamer/window composition)**.** *Considering streamer/window compositions respecting the property 3. If α is such that the window sequence contains the last batch then the following equivalences stand:*

$$\begin{aligned} S &\equiv I_S(S[\alpha, i+k, 1n]) \\ &\equiv I_S(S[B]) \equiv I_S(S[\infty]) \\ &\equiv R_S^u(S[B]) \end{aligned}$$

[10] [4, 32] present also results on this topic.

Example 12: *Timestamp transmission illustration*

Let us consider the stream $S(Id, Value, t)$ of figure 2 and the windows $S[B]$. In this case, the property 3 insure that $I_S(S[B]) = S$. The insertion of the tuple in the stream take place at the time equal to the timestamp of the tuple. Therefore the I_S operator will stamp the tuple at this time. Hereafter are showed the sequence of states that the temporal relation $S[B](t)$ will go through and the corresponding resulting streams $I_S(S[B])$:

$$S = (1, v1, 3); (2, v2, 9); (1, v3, 10); (3, v4, 12); ...$$

$S[B](3)$:

Id	$Val.$	t
1	$v1$	3

$S[B](9)$:

Id	$Val.$	t
2	$v2$	9

$S[B](10)$:

Id	$Val.$	t
1	$v3$	10

$I_S(S[B])$:

Id	$Val.$	t
1	$v1$	3
2	$v2$	9
1	$v3$	10
...	...	...

Example 13: *Timestamp transmission counter-example*
The *timestamp transmission* property is not true for all kind of windows. Let consider the same stream S and the window $S[is, is + 2s, 2s] \equiv$ *range* $2s$ *every* 2 *second* which we will note $S[R_{2s}]$. Then $I_S(S[R_{2s}]) \neq S$. The insertion in the resulting stream is ruled by window changes, therefore the timestamp of generated tuples is the time when the content of the window changes and not the original timestamp. Hereafter are showed the sequence of states that the temporal relation $S[R_{2s}](t)$ will go through and the corresponding resulting streams $I_S(S[R_{2s}])$:

$$S = (1, v1, 3); (2, v2, 9); (1, v3, 10); (3, v4, 12); ...$$

$S[R_{2s}](4)$:

Id	$Val.$	t
1	$v1$	3

$S[R_{2s}](10)$:

Id	$Val.$	t
2	$v2$	9
1	$v3$	10

$S[R_{2s}](12)$:

Id	$Val.$	t
3	$v4$	12

$I_S(S[R_{2s}])$:

Id	$Val.$	t
1	$v1$	4
2	$v2$	10
1	$v3$	10
3	$v4$	12
...	...	...

6.2. Associativity and commutativity

The associativity and the commutativity form the basis of the relational algebraic optimizer inside database systems. For instance, knowing that we can manage to move the selections and projection in a query tree towards the leaves, we can then shrink the different internal data size and rate. Therefore, it is fundamental to look for query equivalences on the different operator and on their influence on each other.

Let's first consider the associativity between the window sequence operator and projection or renaming operator.

Property 4 (Commutativity projection, renaming - windows)**.** *Let S be a stream and α, β, r be a window sequence description (WSD), then, the following properties state:*

$$\Pi_{a_1,...,a_k,t} S[\alpha, \beta, r] \equiv (\Pi_{a_1,...,a_k,t} S)[\alpha, \beta, r]$$

$$\rho_{b/a} S[\alpha, \beta, r] \equiv (\rho_{b/a} S)[\alpha, \beta, r]$$

These results hold because the projection and the renaming operators do not affect any of the elements (i.e., timestamps or tuple positions) used for window creation even with the use of τ_S. It is not always the case for the selection operator. Associativity for selection holds for full temporal windows but only for some specific cases of positional windows.

Property 5 (Associativity selection - temporal windows)**.** *Let S be a stream, α, β, r be a full temporal WSD and c be a selection condition, then, the following property states:*

$$\sigma_c S[\alpha, \beta, r] \equiv (\sigma_c S)[\alpha, \beta, r]$$

This associativity holds because the selection eliminates tuples but does not change the criteria on time used for window creation. For positional windows, associativity is true for only a particular subset of cases. Let's show an example where it doesn't hold.

Example 14: Let S be a stream of temperatures with schema $(temp, t)$. Consider the following queries

1. $\sigma_{temp>0}(S[i, i + 9, 1n])$ is *the set of positive temperatures selected from the last ten measurements each time a measurement is received*
2. $(\sigma_{temp>0} S)[i, i + 9, 1n]$ means *the set of the last ten positive measurements each time a positive measurement is received.*

The width of the resulting windows of these queries are different. The second query creates windows of 10 tuples with positive temperatures whereas in the first one, the resulting windows may contain less than 10 items. This is because the selection of positive temperatures is made on the windows containing 10 tuples with any temperature.

The problem arises because $\tau_{\sigma_c S}$ and τ_S are different. This implies subtile, but meaningful differences in the results. So associativity is globally false for any positional

windows. Yet, in the particular case of a (classical) landmark window sequence, the property is true. The complete proof of these results are available in appendix A.

Property 6 (Associativity selection - cumulative positional windows)**.** *The associativity of selection and positional windows holds for unbounded accumulative window sequence* $[\infty] = [0, i, 1n]$.

Property 6 is true because the unbounded window sequence can be defined as follows, without using the τ_S function: $S[\infty](t,i) = \{s \in S / (t_0, 0) \leq \mathcal{B}_S(s) \leq (t,i)\}$.

These three last properties illustrate how to reduce the internal relation size (by selecting and projecting) without loosing any information, which has direct consequences on the performances of an implementation.

Properties 2 and 5 where already stated in [6] and we prove them in the Astral context.

To go on with the results on associativity, the Codd's unary operators (that is to say, renaming, projection and selection) commute quite easily with stream creators.

Property 7 (Commutativity unary operators - streamer)**.** *Let R be a relation, let c be a selection condition not on t and let a be a list of attributes. The following equalities stand:*

$$\sigma_c I_S(R) \equiv I_S \sigma_c(R) \qquad \Pi_a I_S(R) \equiv I_S \Pi_a(R)$$
$$\sigma_c D_S(R) \equiv D_S \sigma_c(R) \qquad \Pi_a D_S(R) \equiv D_S \Pi_a(R)$$
$$\sigma_c R^r_S(R) \equiv R^r_S \sigma_c(R) \qquad \Pi_a R^r_S(R) \equiv R^r_S \Pi_a(R)$$

Such equalities also stand for renaming.

Moreover, the following equality states if $\forall (t,i)$, $R(t,i) \cap R((t,i)^-) = \emptyset$:

$$\sigma_c R^u_S(R) \equiv R^u_S \sigma_c(R) \qquad \Pi_a R^u_S(R) \equiv R^u_S \Pi_a(R)$$
$$\rho_{y/x} R^u_S(R) \equiv R^u_S \rho_{y/x}(R)$$

There is a strong link between the stream-to-stream unary operators and relation-to-relation unary operator. Thanks to the *timestamp transmission* and the corollary 2, we can see the stream as a composition. As a consequence, it is possible to describe the three basic stream unary operators as a composition of a window, a streamer and a relational unary operator.

Corollary 3 (Selection on stream as a composition)**.** *Let S be a stream and c be a selection condition on S.*

$$\sigma_c S \equiv I_S(\sigma_c S[B]) \equiv I_S(\sigma_c S[\infty]) \equiv R^u_S(\sigma_c S[B])$$

The analogous property stands for projection and renaming.

This corollary shows how the usual implementation works. In practice, in order to perform a selection, usually, the processor fetch the last tuple(s) of the stream. It will make an operation (select for instance) and then will send the result in a new stream. We justify here that the theory can illustrate such behaviour. This property also justfies the equivalence of our modelisation of the three basic stream→stream operator compared to the usual composition definition.

Given these properties for the core operators, properties on the composite operators can be stated.

Corollary 4 (Commutativity stream join - selection)**.** *Let S and S' be two streams. Let c be a selection condition on S and d be a temporal interval.*

The following equality stands:

$$\sigma_c(S \overset{d}{\bowtie} S') \equiv (\sigma_c S) \overset{d}{\bowtie} S'$$

Moreover, if c is a condition on a relation R then,

$$\sigma_c(S \bowtie R) \equiv S \bowtie (\sigma_c R)$$

Corollary 4 states because the streamer I_S is used by the join operator. This streamer has a good behavior in regards to its associativity. If R^u_S had been chosen (which is also a valid choice) for the join operator, such property would not stand.

Remark 1. *When joining a stream with a relation:*

$$\sigma_{c'}(S \bowtie R) \not\equiv (\sigma_{c'} S) \bowtie R$$

To illustrate, let S be a stream $(id, \sqcup) : S = \{(1,1),(2,2),(3,3)\}$ and R a relation having an attribute id. $R(0) = \{(0)\}$ and $R(2.5) = \{(1)\}$. With this data, $(1,1,2.5) \notin \sigma_{id=1}(S \bowtie_{id} R)$ and $(1,1,2.5) \in (\sigma_{id=1} S) \bowtie_{id} R$.

As seen in Section 4.2 the symmetric property does not stand for cartesian product and join in the general case. Nevertheless, the two following properties show that associativity states under certain conditions.

Property 8 (Associativity on cartesian product)**.** *The cartesian product with the lexicographic order is associative. Let A, B and C be three temporal relations, then:*

$$A \times (B \times C) = (A \times B) \times C$$

Proof sketch: As for the classical cartesian product the content of the result is the same in both cases and only the tuples ordering may differ. Tuples ordering will be the same, if the orderings induced by $((\varphi_A, \varphi_B), \varphi_C)$ and by $(\varphi_A, (\varphi_B, \varphi_C))$ are the same. If the lexicographic order is used, the two results are equivalent as the final order is given by $(\varphi_A, \varphi_B, \varphi_C)$. The complete proof is available on appendix A.12.

Under certain conditions this property can also be stated for stream joins.

Corollary 5 (Associativity on band-join)**.** *The band-join (def. 34) is associative for the case $d = 0$ and $d = \infty$ using the lexicographic order. Let S_1, S_2 and S_3 be three streams, then:*

$$S_1 \overset{d}{\bowtie} (S_2 \overset{d}{\bowtie} S_3) = (S_1 \overset{d}{\bowtie} S_2) \overset{d}{\bowtie} S_3$$

This is a useful result, but unfortunately, it states only for special cases of d.

We now have seen some useful properties that have direct consequences on the implementation as the core properties of the relational algebra. We can now apply the basic optimization heuristics such as the selection and projection as close to the leaves possible.

6.3. Transposition

Operator sharing [4, 21, 37] is an optimization technique known to be effective in data stream querying where the execution of a query may be infinite. For achieving an efficient sharing, the largest query equivalence is required. The exact match of two queries requires the equivalence of the query expression and their starting timestamps. The use of windows in the queries introduces different cases.

This section presents a contribution to compare two uniform[11] window descriptions namely, $[\alpha, \beta, r]$ and $[\alpha', \beta', r]$ with two different starting timestamp t_0 and t_1. The objective is to facilitate the sharing of window creation operators even if the two windows are not identical.

- We shall note the two queries as $Q_1 = (S[\alpha, \beta, r], t_0)$ and $Q_2 = (S[\alpha', \beta', r], t_1)$
- D is the implicit slide between two window sequence descriptions
  - For full temporal description: $D = 0$
  - For full positional description: $D = \tau^{-1}_{(S,t_0)}((t_1, 0)^-) + 1$
- B_t is the minimal value of a lower bound of a description starting at t
  - If using temporal bounds: $B_t = t$
  - If using positional bounds: $B_t = 0$
- K is the synchronization factor: $K = \frac{\beta'(0) - \beta(0) + D}{r}$ It states the number of windows that were evaluated for Q_1 when the first window is evaluated for Q_2.

In the following, we make the basic hypothesis that two queries using the same stream S at the same time, see the same data even if the query evaluations started at different moments.

Property 9 (Linear Window Transposing)**.** *Let's consider a stream S and queries $(S[\alpha, \beta, r], t_0)$ and $(S[\alpha', \beta', r], t_1)$. If the WSD have the following form:*

$$\alpha(i) = \max(ai + b, B_{t_0}) \quad \alpha'(i) = \max(ai + b', B_{t_1})$$
$$\beta(i) = ci + d \qquad \beta'(i) = ci + d'$$

If, the following properties state:

$$\begin{aligned} K &= \frac{d' - d + D}{r} \in \mathbb{N} \\ b' &= b + aK - D \text{ if } a \neq 0 \\ \max(B_{t_1}, b') &= \max(B_{t_1}, b - D) \text{ if } a = 0 \\ c &= r \text{ if } K \neq 0 \end{aligned}$$

Then, the following equivalence states:

$$(\sigma_{t \geq t_1} S[\alpha, \beta, r], t_0) \equiv (S[\alpha', \beta', r], t_1)$$

Having K as an integer is rather intuitive as it describes the synchronization between the queries. In this case, it is possible to prove that

$$\gamma'(t, i) = \gamma(t, i) - K.$$

Considering this aspect, by replacing γ with γ', and by limiting the lower bound to $t \geq t_1$, the conditions over b' can be found.

The property about c states that the upper bound must follow the evaluation. This is necessary because the upper bound of the first window is always the moment of the first evaluation.

The complete demonstration is available on appendix B. This property is actually a corollary of a more general theorem that does not consider a linear description (see appendix B).

Example 15: Let's consider queries:

$$Q_1 = (S[2is + 3s, 5is + 3s, 5s], t_0)$$
$$Q_2 = (S[2is + 2s, 5is + 8s, 5s], t_1)$$

Q_2 is launched after Q_1 ($t_0 < t_1$). The objective is to see if it is possible to reuse the results of Q1 for Q2. Here:

$$K = \frac{(8s + t_1) - (3s + t_0)}{5s} = 1 + \frac{t_1 - t_0}{5s}.$$

To be able to reuse the results, t_1 has to be $t_1 = t_0 + n * 5s$.

As $K \neq 0$, c has to be equal to the rate. As $a \neq 0$, b' must be equal to $3s + t_0 + 2s * K = 5s + t_0 + n * 2s$. Therefore, we have, $b' - t_0 = 5s - n * 3s$. The problem is then reduced to one single equation to verify if the two windows can be coherent. For our example, $2 = 5 - 3n \Rightarrow n = 1$ is a valid matching. The figure 5 represents the result of this transposing.

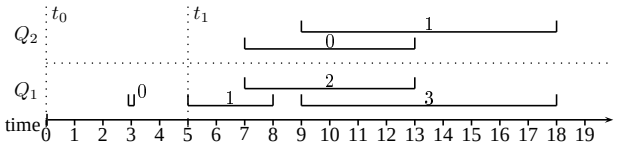


Figure 5: Synchronized windows for operator sharing

Property 9 shows a result for windows with linear descriptions. Such kind of windows are commonly used. This result can be integrated in stream query evaluators as the proposed equalities can be verified easily at runtime.

As another corollary, the following property points out the case of two queries using window sequences following the same pattern but having an initial offset.

[11] the bounds and the rate are of the same type

Corollary 6 (Linear Natural Transposing)**.** *Let S be a stream and consider queries $(S[\alpha, \beta, r], t_0)$ and $(S[\alpha', \beta', r], t_1)$. If the WSD have the following form:*

$$\begin{array}{rl} \alpha(i) - B_{t_0} = \alpha'(i) - B_{t_1} = & \max(ai + b, 0) \\ \beta(i) - B_{t_0} = \beta'(i) - B_{t_1} = & ci + d \end{array}$$

If $c = r \wedge (a = 0 \wedge b \leq 0 \vee a = r)$ and

$$\begin{array}{rcll} t_1 - t_0 & \in & r\mathbb{N} & \textit{in temporal} \\ \tau^{-1}_{(S,t_0)}(t_1) & \in & r\mathbb{N} & \textit{in positional} \end{array}$$

Then, the following equivalence states:

$$(\sigma_{t \geq t_1} S[\alpha, \beta, r], t_0) \equiv (S[\alpha', \beta', r], t_1)$$

The conditions on the window synchronization are simplified. This allows us to state that usual window sequences, as sliding windows and landmark windows, are naturally *transposable*.

Sliding window sequence are similar to $[ir, ir + w, r]$ which verifies the criterion. Therefore, a description $[5is, 5is + 2s, 5s]$ can be transposed by an interval of any multiple of the rate r.

To our knowledge, a generic equivalence of windowing operators over time were never investigated this far. Further details on this theory, as well as demonstrations of this section, are available on the Astral's wiki [34].

7. Astral and related work

Section 2 presented the motivation of this work and discussed existing sensor querying solutions. This section provides complementary discussions: Section 7.1 summarizes Astral querying capabilities with examples and section 7.2 discusses specific points of our proposal with respect to related work.

7.1. Astral Querying Capabilities

This work was motivated by offering a formal support for a large variety of queries useful in sensor based applications. Astral can be used to formalize all kinds of queries presented in section 2.2. Let's consider here some examples.

- Present instantaneous queries. "Which buoys are presently (t_q) in zone '7016' ":

$$\begin{array}{l} ((\Pi_{BuoyId}\sigma_{Loc=7016}AnchBuoys) \cup \\ \quad (\Pi_{BuoyId} \\ \qquad (\sigma_{value=7016} S \bowtie \sigma_{type=GPS} Sensors)[id/L]) \\ )(t_q) \end{array}$$

- Past and present instantaneous queries. "Presently (t_q), which is the last buoy being in zone '7016' ?". Stating that the history of the stream is the relation $H = (S \bowtie Sensors)[\infty]$ started at $t_h \ll t_q$:

$$\begin{array}{l} (\\ \quad (\Pi_{BuoyId}\sigma_{Loc=7016}AnchBuoys) \cup \\ \quad (\Pi_{BuoyId}\sigma_{value=7016 \wedge type=GPS} I_S(H))[id/L] \\ )(t_q) \end{array}$$

- Continuous query using an instantaneous queries for designation. "Data from SST sensors currently measuring more than 25°C at start time (t_0)":

$$S \ltimes \Pi_{id}\sigma_{type='SST' \wedge value \geq '25'}(S \bowtie Sensors)[id/L]^{t_0}$$

- Continuous query using a continuous queries for designation. "Data from SST sensors measuring more than 25°C":

$$S \ltimes \Pi_{id}\sigma_{type='SST' \wedge value \geq '25'}(S \bowtie Sensors)[id/L]$$

It can be noticed that there exists common patterns like the window $[id/L]$ or a join between the stream S and the meta-data relations. Further works could investigate on the design of the usual query patterns in the field of sensor data querying. We could imagine an efficient generic data representation and a designation operator for instance.

7.2. Related work

The preceding sections already provided comments on choices made in Astral with respect to related work. This section presents discussions on the main aspects. A complete presentation of the referenced systems is out of the scope of this paper.

It is worth noting that several systems mentioned hereafter do not provide complete formal definitions. The comparison of such works and our formal proposal is not straightforward.

Aspect	time
Astral	$t \in \mathcal{T} \sim \mathbb{R}$
Description	continuous time

This choice allows a precise specification of the time avoiding heterogeneity and implicit dependency of the implementation. Note that using discrete time would lead semantics of operators to depend on this discretization and therefore, on the implementation. For example STREAM [4] defines the operator $\mathcal{RS}$ with the statement *for each timestamp*. For one implementation, the timestamp width could be *1ms* and for an other it could be *1s*. Many systems like Aurora [2] does not have restrictions on the timestamp type but the set of supported operators is restricted to stream-to-stream operations. Our choice allows to provide a wide range of operators with a consistent time handling.

Aspect	Abstract model
Astral	$s \in R(t, i)$, $s \in S$, $\mathcal{B}_S$
Description	Two main concepts: temporal relation and stream

Astral's choices are inspired by the two-fold approach of STREAM and CQL [6] which is nowadays the most widely used. Others, like SStreaMWare [18] extends the SEQ's model [38] by proposing the concept of window sequences (list of window states) which are similar to temporal relations created by Astral and CQL windows. Aurora only cares of streams but with more limited expressivity (no generic join for example). Batches are rarely used as the notion is new. Few clear formalizations [8] propose adding another special attribute *bid* to each tuple in order to distinguish the batches. Some formalizations [8, 32] do not use the term "*relation*" but the notion of "*stream window*". This notion is close to a subset of $S[\infty]$.

Aspect	Timestamp Order
Astral	$\mathcal{B}_S$ increasing function
Description	For a stream, its positional order and the order inferred by timestamps is coherent.

The consistency of the positional and the timestamp orders is a postulate in Astral. In SQuAl, the algebra of Aurora, this assumption is not made. To insure the consistency of the model, they require an operator *Order*. In practice, the implementation has to fulfill this property. Interesting works [28] are made to ensure the consistency of the positional and timestamp orders without breaking the workflow. As a matter of fact, Oracle [44], *advises* for performance reasons, to fulfill this property in order to avoid complex treatments that would be otherwise required to obtain consistent results. We claim that this property must be verified to have a coherent algebra.

Aspect	Binary relational operators
Astral	$R_1 \times R_2$, $R_1 \cup R_2$... with Φ
Description	Operators definition include the definition of a strict ordering of the results. An unspecified function is used for that purpose

The relational part of a stream algebra has mainly be considered in the literature as a direct application of the relational algebra. Our formal framework shows that the use of physical identifiers forces to define the final ordering of the tuples produced by the operators. We also showed problems that can appear (see property 2) as the ordering is not symmetric in the general case for those operators. This is important because symmetry is used in several optimization approaches in relational DBMS. To the best of our knowledge, such formalization and results have never been presented.

Aspect	Streamers relation→stream
Astral	I_S, D_S, R^r_S, R^u_S
Description	Operators producing streams reflecting updates on a relation or periodic selections on a relation.

The introduction of steamers allows a precise specification of the relation - stream "transformation". This kind of operators are not clearly supported by all current proposals as it relies on a separation of the notions of relation and stream. The notion of streamer is implicit in some proposals. For example Borealis [1] uses *messages* to detect inserts and updates in a relation. STREAM does introduce streamers ($\mathcal{IS}$, $\mathcal{DS}$,$\mathcal{RS}$) using an implicit *chronon*[12] r. R^r_S in Astral is similar to $\mathcal{RS}$ in STREAM. Streamers in Astral are well integrated with the other operators and provide flexibility and precision for query expression.

Aspect	Windows
Astral	$[\alpha, \beta, r]$
Description	A window is defined by two functions for boundaries evolution and a rate of evaluation.

Window creation may take numerous semantics. Many works on formalization and implementation exist. In a previous paper [35], we showed how our model covers all the usual definitions (range/slide/row, i.e. sliding, tumbling and landmark windows positional or temporal) and how it can express window sequences offered by procedural approaches as the earlier versions of TelegraphCQ [10] (which uses a for-loop). We also showed the formalization of cross-domain WSD like "*Get the last 20 tuples each 2min*" which have got little attention but are often required in practice.

Some formal works exist. Among them [32] which uses a mathematical set-based approach allowing formal proofs on properties. In this approach definitions are very long and each type of window has to be detailed from the beginning. Astral includes definitions using only 3 parameters (α, β, γ) but providing nevertheless high flexibility and allowing the expression of all kind of window sequences reported in literature. SECRET [8] formalizes the window operator from an execution point of view. It is based on 4 main concepts which can be found in Astral. The *Tick* decides when the system must react to the input (tuple-based, time-based or batch-based), the *Content* is defined from the input and a *Scope*. A *Report* sends the result to the next part of the query. Although their approach is orthogonal to ours, we have similar notions: the scope is modeled by α, β and the ticks and report are made by γ. The approach of SECRET is execution-based and it is easy to describe the behavior of different execution systems but very hard to prove properties on it.

Aspect	Stream Join
Astral	$S \bowtie R$, $S \bowtie^d S'$, $S \ltimes R$
Description	Joining streams or relation with stream

[12] Chronon of a system implementation is the smallest timestamp difference that can be obtained (e.g. the chronon of a unix-timestamped system is 1ms)

Joining streams may be done in several ways. Each system/model uses a different form for the join operator. STREAM support relies on the relational join. Stream join operator in Aurora is based on timestamp correspondences. GSN [3] uses a positional correspondence between streams. SStreaMWare also proposed a join operator based on symmetric hash join [43]. Chronicles [23] introduces a "natural" join using timestamp matching. As in STREAM, we consider that a generic join operator does not exist as there are too many possible useful semantics. Although, we defined three operators that can easily cover usual joins like the ones mentioned earlier. We can also see that the formalization of the domain manipulation allows the definition of the semi-sensitive join which is a rather implementation oriented operator.

Aspect	Domain manipulation
Astral	D_c^f, R^{t_s}
Description	A temporal relation can be defined by past states of other relations

This operation is often used but have not been formalized before. Its formalization allows the expression of other operators (e.g. time fixation, definition 22, and semi-sensitive join, definition 33) also quite common in practice. The formalization in Astral is simple.

Aspect	Spread Operator
Astral	$\lhd_{a_1,\ldots,a_n}$, $\rhd_{a_1,\ldots,a_n}$
Description	Refines or recreates batches

The spread operator has been introduced in [24] but was never included in a complete algebra. Our formalization allows an easy manipulation of batches. The spread operation is equivalent to a redefinition of $\mathcal{B}_S$. Astral provides a good integration of such concepts.

Astral extends current proposals and allows the formal expression of a large variety of queries useful in sensor based systems. This proposition also allows to point out, and to handle, semantic problems that were not yet identified (or implicit) in existing proposals. Operations that are very "practice-oriented" can also be formalized with our proposal.

8. Implementation

As a proof of concept, the Astral algebra has been implemented in a prototype[13]. It implements all the concepts presented in this paper. For instance, the implementation of the Window Sequence Operator, rely on the τ_S, τ_S^{-1} and $\mathcal{B}_S$ functions which are directly implemented in the stream class.

We have considered some light restrictions with respect to the pure theory. For instance, in the window sequence descriptions α and β are increasing (non strictly). This assertion is common and all examples presented in this paper verify it. This point is important from the implementation point of view as it allows deletion of a tuple that goes out of the buffer and facilitates memory management. In theory, such non-increasing function is still possible, nevertheless some properties like prop.3 needs it.

Our prototype has a basic scheduler which drives the execution of operators. Some aspects described in the algebra can be directly seen in the prototype. For instance, operators like sensible streamers are executed when they observe changes on the input relation. In the case of independent streamers, the operator directly tells to the scheduler when it must be executed.

As most stream management systems, the source stream is used as a regulation of the time with its timestamp. Hence, the problem of network latency is not directly troublesome as we consider the time indicated by the timestamp attribute. In order to proceed without tuple loss, we must wait for the next tuple to see the current time. As used in many sensor network projects and large-scale distributed systems, heartbeat messages would solve the problem of unbounded waiting.

The prototype is coded in *Java* using *OSGi*/*iPOJO*[14] technologies over the *Apache Felix* framework. From an architectural point of view, each operator, and each stream (or relation) is a component. Scheduler, query runtime and builders are services that form the Astral core, which can execute the whole life-cycle of a query. The project's core requires more than 8 kSLOC (without comments).

Distributed continuous query processing on several nodes is possible with this prototype. Two special operators has been defined for this purpose, an emitter and a receiver entity. By creating queries using those operators as source/head on multiple instances, it is possible to distribute query evaluation on many nodes. The implementation of the operators synchronizes the two schedulers in order to have the exact behavior described by the algebra.

A query may involve several streams or relations denoted as data sources. The execution plan of a query is generated using Astral operators. The only optimization implemented in this version is a rather simple "push-projection"-like strategy. We look forward for a more complete optimization process using properties exposed in this article.

9. Conclusion and future work

This paper proposed a formal framework to express both continuous and instantaneous queries on sensor data represented as streams and temporal relations. It allows a precise expression of the semantics of the operators. This is important for a better understanding of query semantics implemented by these systems and in potential mediation between them.

The results presented in this paper extend existing formal work by covering a more complete definition of the set

[13] http://astral.ligforge.imag.fr

of operators including stream creation, Cartesian product and join on streams and/or relations and a general window creation operator which provides full flexibility to define temporal, positional and hybrid windows.

This paper also demonstrated several properties of the proposed operators. The associativity of some operators is presented and in addition some particular cases where associativity does not hold are discussed. Transposition of windows is also described. These properties are particularly useful in query optimization for query rewriting and re-ordering operators in a query plan. The proposed results will also facilitate sharing operators across concurrent queries which is a promising approach (as shown in [4, 11]). All these are contributions to establish the bases required for query optimization in distributed sensor data management.

The core of this research is the formal proposal but experimental work has also been accomplished in parallel. An Astral prototype has been developed in Java as a proof of concept. A service oriented approach has been adopted. The implementation uses a OSGi framework. This choice allows dynamic discovering of devices producing heterogenous data streams. Queries can operate on sensors or other type of devices.

Future research is planned to extend both, the theoretical and the practical work. On the theoretical side the research agenda includes further work on the operator's properties, on query equivalence and query rewriting. More globally efforts are required on the combination of several optimization technics as in-network aggregation [27, 29], load shedding[41] and logical optimization to build efficient and rich sensor data query processors.

Acknowledgment: Many thanks to Solveig Albrand for her help with this paper. This research is supported by the Transcap and ALAP projects.

References

[1] Abadi, D., Ahmad, Y., Balazinska, M., gur Çetintemel, U., Jan. 2005. The Design of the Borealis Stream Processing Engine. CIDR '05: Proceedings of the Second Biennial Conference on Innovative Data Systems Research.

[2] Abadi, D., Carney, D., gur Çetintemel, U., Cherniack, M., Convey, C., Lee, S., Stonebraker, M., Tatbul, N., Zdonik, S., 2003. Aurora: a new model and architecture for data stream management. VLDB '03: Proceedings of the 29th international conference on Very large data bases 12 (2).

[3] Aberer, K., Hauswirth, M., Salehi, A., 2007. Infrastructure for Data Processing in Large-Scale Interconnected Sensor Networks. In: MDM '07: International Conference on Mobile Data Management 2007. pp. 198–205.

[4] Arasu, A., Babcock, B., Babu, S., Cieslewicz, J., Datar, M., Jan. 2004. STREAM: The Stanford Data Stream Management System. Data Stream Management: Processing High-Speed Data Streams.

[5] Arasu, A., Babu, S., Widom, J., 2002. An Abstract Semantics and Concrete Language for Continuous Queries over Streams and Relations.

[6] Arasu, A., Babu, S., Widom, J., 2006. The CQL continuous query language: semantic foundations and query execution. VLDB '06: Proceedings of the 32nd international conference on Very Large Data bases 15 (2).

[7] Babcock, B., Babu, S., Datar, M., Motwani, R., Widom, J., Mar. 2002. Models and Issues in Data Stream Systems. Tech. Rep. 2002-19, Stanford University.

[8] Botan, I., Derakhshan, R., Dindar, N., 2010. SECRET: A Model for Analysis of the Execution Semantics of Stream Processing Systems. In: VLDB '10: Proceedings of the 36th international conference on Very Large Data bases. pp. 232–243.

[9] Carney, D., gur Çetintemel, U., Cherniack, M., Convey, C., Jan. 2002. Monitoring streams: a new class of data management applications. VLDB '02: Proceedings of the 29th international conference on Very large data bases.

[10] Chandrasekaran, S., Cooper, O., Deshpande, A., Jan. 2003. TelegraphCQ: Continuous dataflow processing for an uncertain world. CIDR '03: Proceedings of the First Biennial Conference on Innovative Data Systems Research.

[11] Chen, J., DeWitt, D., Tian, F., Wang, Y., 2000. NiagaraCQ: a scalable continuous query system for Internet databases. SIGMOD '00: Proceedings of the 2000 ACM SIGMOD international conference on Management of data.

[12] Codd, E. F., 1970. A relational model of data for large shared data banks. Communications of the ACM 13 (6).

[13] Cranor, C., Johnson, T., Spataschek, O., Shkapenyuk, V., 2003. Gigascope: a stream database for network applications. SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data.

[14] Escoffier, C., Bourcier, J., Lalanda, P., Yu, J., 2008. Towards a Home Application Server. CCNC 2008: 5th IEEE Consumer Communications and Networking Conference, 2008, 321–325.

[15] Franklin, M. J., Jeffery, S. R., Krishnamurthy, S., Reiss, F., Rizvi, S., Wu, E., Cooper, O., Edakkunni, A., Hong, W., Jan. 2005. Design considerations for high fan-in systems: The HiFi approach. CIDR '05: Proceedings of the Second Biennial Conference on Innovative Data Systems Research.

[16] Golab, L., Özsu, M. T., 2005. Update-pattern-aware modeling and processing of continuous queries. SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data, 658.

[17] Gripay, Y., Laforest, F., Petit, J.-M., 2010. A simple (yet powerful) algebra for pervasive environments. In: EDBT '10: Proceedings of the 13th International Conference on Extending Database Technology. ACM Press, New York, New York, USA, p. 359.

[18] Gürgen, L., Jan. 2007. Gestion à grande échelle de données de capteurs hétérogènes. PhD Thesis, INP Grenoble.

[19] Gürgen, L., Roncancio, C., Labbé, C., 2010. Data Management Solutions in Networked Sensing Systems. Wireless Sensor Network Technologies for the Information Explosion Era 278, 111–137.

[20] Hammad, M., Aref, W., Franklin, M., Mokbel, M., Elmagarmid, A., 2003. Efficient execution of sliding window queries over data streams.

[21] Hong, M., Riedewald, M., Koch, C., Gehrke, J., Demers, A., 2009. Rule-based multi-query optimization. In: EDBT '09: Proceedings of the 12th International Conference on Extending Database Technology Advances in Database Technology - EDBT '09. ACM Press, New York, New York, USA, p. 120.

[22] Inc., S., 2010. SQLstream. `http://www.sqlstream.com`.

[23] Jagadish, H., Mumick, I., Silberschatz, A., 1995. View maintenance issues for the chronicle data model (extended abstract). PODS '95: Proceedings of the 14th ACM symposium on Principles of database systems.

[24] Jain, N., Mishra, S., Srinivasan, A., Gehrke, J., Widom, J., Balakrishnan, H., gur Çetintemel, U., Cherniack, M., Tibbetts, R., Zdonik, S., 2008. Towards a streaming SQL standard. VLDB '08: Proceedings of the 34th international conference on Very Large Data bases 1 (2), 1379–1390.

[25] Johnson, D. R., Garcia, H. E., Boyer, T. P., July 2006. World ocean database 2005 tutorial. Tech. rep., Ocean Climate Laboratory, National Oceanographic Data Center, Silver Spring, Maryland.

[26] Jurdak, R., Nafaa, A., Barbirato, A., Nov. 2008. Large Scale

Environmental Monitoring through Integration of Sensor and Mesh Networks. MDPI Sensors: Molecular Diversity Preservation International Sensors.
[27] Jurdak, R., Ruzzelli, A. G., Barbirato, A., Boivineau, S., Mar. 2009. Octopus: Modular Visualization and Control for Sensor Networks. Wiley Wireless Communications and Mobile Computing.
[28] Krishnamurthy, S., Franklin, M., Davis, J., Farina, D., Golovko, P., Li, A., Thombre, N., 2010. Continuous analytics over discontinuous streams. In: SIGMOD '10: Proceedings of the 2010 ACM SIGMOD international conference on Management of data. ACM, pp. 1081–1092.
[29] Madden, S. R., Franklin, M. J., Hellerstein, J. M., Hong, W., 2002. TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks. OSDI 2002: 5th Symposium on Operating System Design and Implementation.
[30] Madden, S. R., Franklin, M. J., Hellerstein, J. M., Hong, W., Mar. 2005. TinyDB: an acquisitional query processing system for sensor networks. TODS '05: Transactions on Database Systems 2005 30 (1), 122–173.
[31] Oceanographic, A., Laboratory, M., 2010. The global drifter program.
URL \url{http://www.aoml.noaa.gov/phod/dac/index.php}
[32] Patroumpas, K., Sellis, T. K., 2006. Window specification over data streams. ICSNW'06: Proceedings of the International Conference on Semantics of a Networked World 4254, 445–464.
[33] Pazan, S. E., Niiler, P., January 2004. New global drifter data set available. Eos, Transactions, American Geophysical Union 95 (3).
URL http://www.agu.org/eos_elec/000440e.shtml
[34] Petit, L., 2010. Astral: Advanced Stream Algebra, Wiki. http://sigma.imag.fr/astral.
[35] Petit, L., Labbé, C., Roncancio, C. L., 2010. An Algebric Window Model for Data Stream Management. In: MobiDE '10: Proceedings of the 9th International ACM Workshop on Data Engineering for Wireless and Mobile Access. ACM, pp. 17–24.
[36] Reiss, F., Stockinger, K., Wu, K., Shoshani, A., Hellerstein, J. M., 2007. Enabling Real-Time Querying of Live and Historical Stream Data. In: SSDM' 07: Proceedings of the 19th International Conference on Scientific and Statistical Database Management. p. 28.
[37] Sellis, T. K., 1988. Multiple-query optimization. TODS '88: ACM Transactions on Database Systems 13 (1), 23–52.
[38] Seshadri, P., Livny, M., Ramakrishnan, R., Jan. 1995. SEQ: A model for sequence databases. Data Engineering: Bulletin of the Technical Committe on Data Engineering 2003.
[39] StreamBase Systems, I., 2009. StreamSQL online documentation. http://streambase.com/developers/docs/latest/streamsql/index.html.
[40] Sullivan, M., Sep. 1996. Tribeca: A Stream Database Manager for Network Traffic Analysis. VLDB '96: Proceedings of the 22th International Conference on Very Large Data Bases.
[41] Tatbul, N., gur Çetintemel, U., Zdonik, S., Cherniack, M., Jan. 2003. Load Shedding on Data Streams. MPDS '03: Proceedings of the ACM Workshop on Management and Processing of Data Streams.
[42] Terry, D., Goldberg, D., Nichols, D., Oki, B., Jun. 1992. Continuous queries over append-only databases. SIGMOD '92: Proceedings of the 1992 ACM SIGMOD international conference on Management of data 21 (2), 321–330.
[43] Wilschut, A. N., Apers, P. M. G., Jan. 1991. Dataflow query execution in a parallel main-memory environment. PDIS '91: Proceedings of the first international conference on Parallel and distributed information systems 1 (1), 68–77.
[44] Witkowski, A., Bellamkonda, S., Li, H.-G., Liang, V., Jan. 2007. Continuous queries in oracle. VLDB '07: Proceedings of the 33rd international conference on Very Large Data bases.
[45] Yao, Y., Gehrke, J., 2002. The cougar approach to in-network query processing in sensor networks. SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of Data 31 (3), 9–18.

Appendices

A. Proofs details

A.1. Prop.1: τ_S

As all timestamps are at least greater than the base timestamp. Then, the first position has a timestamp greater than t_0. Hence $\tau_S(0) \geq (t_0, 0)$. Considering hypothesis 1, τ_S is increasing, thus $\forall n \in P_S, \tau_S(n) \geq (t_0, 0)$.

Let $(t, i) \in \mathcal{T} \times \mathbb{N} \geq (t_0, 0)$. Considering the formal definition of τ_S^{-1}.

- If $|S| < +\infty \wedge (t, i) \geq \tau_S(|S| - 1)$, then we can write $\tau_S(\tau_S^{-1}(t, i)) = \tau_S(|S| - 1) \leq (t, i)$.
- Else : $\tau_S^{-1}(t, i) = \sum_{n=-1}^{|S|-2} n \ \mathbf{1}_{[\tau_S(n), \tau_S(n+1)[}(t, i)$
  Let j be the one (and only) indice such that $(t, i) \in [\tau_S(j), \tau_S(j+1)[$. Then $\tau_S(\tau_S^{-1}(t, i)) = \tau_S(j) \leq (t, i)$.

If $\exists s \in S$ such that $\mathcal{B}_S(s) = (t, i)$, then $\exists j \in P_S$ such that $\tau_S(j) = (t, i)$. Considering previous result, the result is clear.

Finally, let $n \in P_S$, as τ_S is increasing non strictly, there can be equal batches. Let a and b be the largest interval such that $\forall j \in [\![a, b]\!]$, $\tau_S(j) = \tau_S(n)$, by definition $n \in [a, b]$.

$$\begin{aligned}
\tau_S^{-1}(\tau_S(j)) &= \sum_{n=-1}^{|S|-2} n \ \mathbf{1}_{[\tau_S(n), \tau_S(n+1)[}(\tau_S(j)) \\
&= \sum_{n=a}^{b} n \ \mathbf{1}_{[\tau_S(n), \tau_S(n+1)[}(\tau_S(j)) \\
&= b \ \mathbf{1}_{[\tau_S(b), \tau_S(b+1)[}(\tau_S(j)) \\
&= b
\end{aligned}$$

The third line is true as $[\tau_S(n), \tau_S(n+1)[= \emptyset$ for $n \in [a, b[$. As $b \geq n$, hence the result.

A.2. Prop.3: Timestamp transmission for I_S and R_S^u

In the positional domain, recall that

$$\gamma(t, i) = \left\lfloor \frac{\tau_S^{-1}(t, i) - \beta(0)}{r} \right\rfloor$$

Using the given description, $\gamma(t, i) = \tau_S^{-1}(t, i) - k$.
Let $s \in E_{t,i} \Leftrightarrow s \in S \wedge$

$$\begin{array}{lll}
\tau_S \circ \alpha \circ \gamma(t, i) & \leq \mathcal{B}_S(s) \leq & \tau_S(\beta(\gamma(t, i))) \\
\tau_S \circ \alpha \circ \gamma(t, i) & \leq \mathcal{B}_S(s) \leq & \tau_S(\gamma(t, i)) \\
\tau_S \circ \alpha \circ \gamma(t, i) & \leq \mathcal{B}_S(s) \leq & \tau_S(\tau_S^{-1}(t, i))
\end{array}$$

The last inequality is equivalent to $\tau_S \circ \alpha \circ \gamma(t, i) \leq \mathcal{B}_S(s) \leq (t, i)$. The implication is simple due to prop.1. The reciprocal must be detailled. If $\mathcal{B}_S(s) \leq (t, i)$. It is easy to say that $\forall s \in S$, $\mathcal{B}_S(s) \notin]\tau_S(\tau_S^{-1}(t, i)), (t, i)[$, because if it existed one, $\tau_S(\tau_S^{-1}(t, i))$ would be equal to (t, i) which is absurd. Therefore $\mathcal{B}_S(s) \leq \tau_S(\tau_S^{-1}(t, i))$.

Therefore $S[\alpha,i,1n](t,i) = F_{t,i} =$

$$s \in S, \begin{cases} \tau_S \circ \alpha \circ \gamma(t,i) \leq \mathcal{B}_S(s) \leq (t,i) \\ \underset{E_{t,i}}{\text{rpos}}(s) \leq \tau_S^{-1}(t,i) - \alpha \circ \gamma(t,i) \end{cases}$$

$\Psi_{t,i}(s,t) \in S' \wedge \mathcal{B}_{S'}(\Psi_{t,i}(s,t)) = (t,i)$
$\Leftrightarrow s \in F_{t,i} \wedge s \notin F_{(t,i)^-}$
$\Leftrightarrow s \in S \wedge [...]_1 \leq \mathcal{B}_S(s) \leq (t,i) \wedge \underset{E_{t,i}}{\text{rpos}}(s) \leq [...]_2 \wedge$
$([...]_3 > \mathcal{B}_S(s) \vee \mathcal{B}_S(s) > (t,i)^- \vee \underset{E_{(t,i)^-}}{\text{rpos}}\ (s) > [...]_4)$

As $\tau_S \circ \alpha \circ \gamma(t,i) \leq \mathcal{B}_S(s) \wedge \tau_S \circ \alpha \circ \gamma((t,i)^-) > \mathcal{B}_S(s)$ is impossible because $\tau_S \circ \alpha \circ \gamma$ increasing. As well $\underset{E_{t,i}}{\text{rpos}}(s) \leq \tau_S^{-1}(t,i) - \alpha \circ \gamma(t,i) \wedge \underset{E_{(t,i)^-}}{\text{rpos}}\ (s) > \tau_S^{-1}((t,i)^-) - \alpha \circ \gamma((t,i)^-)$ is also impossible because the number tuples with a higher rank than s obviously increase from $(t,i)^-$ to (t,i). Finally, we have the result :
$\Leftrightarrow s \in F_{t,i} \wedge \mathcal{B}_S(s) > (t,i)^-$
Thus, $\mathcal{B}_S(s) = (t,i)$.■

A.3. Window description $[\infty] = [0,i,1n]$

Considering the beginning of proof A.2. It is easy to imply that,

$$s \in E_{t,i} \Leftrightarrow s \in S \wedge (t_0,0) \leq \mathcal{B}_S(s) \leq (t,i)$$

We must now prove that $E_{t,i} = S[0,i,1n](t,i)$. As, $\beta(\gamma(t,i)) - \alpha(\gamma(t,i)) = \tau_S^{-1}(t,i)$.
Let $s \in E_{t,i}$. As $\tau_S^{-1}(t,i)$ corresponds to the position of the tuple that has the biggest physical identifier in $E_{t,i}$ (due to the definition of τ_S^{-1})

$$\underset{E_{t,i}}{\text{rpos}}(s) = \tau_S^{-1}(t,i) - \underset{E_{t,i}}{\text{pos}}(s)$$

Thus $\tau_S^{-1}(t,i) - \underset{E_{t,i}}{\text{pos}}(s) \leq \tau_S^{-1}(t,i)$ which is true.
Finally, $S[0,i,1n](t,i) = \{s \in S, \mathcal{B}_S(s) \leq (t,i)\}$

A.4. Window description $[B] = [\tau_S^{-1}(\tau_S(i)^-) + 1, i, 1n]$

The same proof as proof A.3 is applied here.

First, it is easy to see using prop.1 that $\tau_S(\gamma(t,i)) = \tau_S(\tau_S^{-1}(t,i)) \leq (t,i)$. By definition, $\exists s \in S$, $\mathcal{B}_S(s) = \tau_S(\tau_S^{-1}(t,i)) = (t',i')$. Thus, $\tau_S^{-1}((t',i')^-)$ is the maximal position of all the tuples that have a batch identifier strictly lower that (t',i'). Thus $\alpha(\gamma(t,i))$ corresponds to the first tuple inside the batch (t',i').■

A.5. Cor.2 Windows inversion

$S \equiv I_S(S[\alpha, i+k, 1n]) \equiv I_S(S[B]) \equiv I_S(S[\infty]) \equiv R_S^u(S[B])$
Considering the proof A.2, we have:
$\Psi_{t,i}(s,t) \in S' \wedge \mathcal{B}_{S'}(\Psi_{t,i}(s,t)) = (t,i)$
$\Leftrightarrow s \in S \wedge \mathcal{B}_S(s) = (t,i) \wedge \underset{E_{t,i}}{\text{rpos}}(s) \leq \tau_S^{-1}(t,i) - \alpha \circ \gamma(t,i)$
Considering that the window always contains the last batch, the last part is always true. Hence the result:
$\Leftrightarrow s \in S \wedge \mathcal{B}_S(s) = (t,i)$

As $\Psi_{t,i}$ conserves all the values (even the timestamp in that case). We must now prove that the order is the same. As $\Psi_{t,i}$ conserves the ordering inside the relation and this order is directly inherited from the input stream, the order is therefore equivalent.

Thus $S \equiv I_S(S[\alpha, i+k, 1n])$. Considering the description of $[\infty]$ and $[B]$ that matches with the hypotheses. $S \equiv I_S(S[B]) \equiv I_S(S[\infty])$. By extension the result on $R_S^u(S[B])$ is also correct.■

A.6. Prop.4 Associativity projection, renaming - windows

Let us prove $\Pi_{a,t}S[\alpha,\beta,r] \equiv (\Pi_{a,t}S)[\alpha,\beta,r]$ in the positional case (the hardest).

Let $s \in \Pi_{a,t}S[\alpha,\beta,r]$, thus $\exists s' \in S[\alpha,\beta,r]$ such that $s'[a,t] = s$.
$s \in \Pi_{a,t}S[\alpha,\beta,r]$

$\Leftrightarrow \quad s' \in S[\alpha,\beta,r] \wedge s'[a,t] = s$
$\Leftrightarrow \quad s' \in E_{t,i} \wedge \underset{E_{t,i}}{\text{rpos}}(s) \leq \beta \circ \gamma(t,i) - \alpha \circ \gamma(t,i) \wedge s'[a,t] = s$

γ, and therefore $E_{t,i}$, depends directly on S. Although, as $\gamma(t,i) = \left\lfloor \frac{\tau_S^{-1}(t,i) - \beta(0)}{r} \right\rfloor$ and the fact that $\tau_S = \tau_{\Pi_{a,t}S}$ ($\Pi_{a,t}$ does not change any position), then γ is not affected by $\Pi_{a,t}$. Therefore $E_{t,i}$ is not affected, at least not more that values deleted.

Hence, the result is:

$\Leftrightarrow \quad s[a,t] \in \Pi_{a,t}E_{t,i} \wedge \underset{E_{t,i}}{\text{rpos}}(s) \leq \beta \circ \gamma(t,i) - \alpha \circ \gamma(t,i)$
$\Leftrightarrow \quad s \in (\Pi_{a,t}S)[\alpha,\beta,r]$

The proof is very similar (but way more simple as τ_S is not present) for the temporal case. The proof for renaming is the same.■

A.7. Prop.5 Associativity selection - temporal windows

Similar to proof A.6, in the temporal case. In this case, σ_c does not influence the behaviour of a temporal windows.
$s \in \sigma_c S[\alpha,\beta,r](t,i)$

$\Leftrightarrow \quad s \in S[\alpha,\beta,r](t,i) \wedge c(s)$
$\Leftrightarrow \quad s \in S \wedge \alpha(\gamma(t,i)) \leq \mathcal{B}_S(s) \leq \beta(\gamma(t,i)) \wedge c(s)$
$\Leftrightarrow \quad s \in \sigma_c S \wedge \alpha(\gamma(t,i)) \leq \mathcal{B}_S(s) \leq \beta(\gamma(t,i))$
$\Leftrightarrow \quad s \in \sigma_c S \wedge \alpha(\gamma(t,i)) \leq \mathcal{B}_{\sigma_c S}(s) \leq \beta(\gamma(t,i))$
$\Leftrightarrow \quad s \in (\sigma_c S)[\alpha,\beta,r](t,i)$

It is possible here as α, β, and γ does not depend on the input S.

In the positional case, we introduce τ_S (inside γ and on the scope condition). As $\tau_{\sigma_c S} \neq \tau_S$, the result tends to be false. Counter examples easily prove it to be wrong due to this difference.■

A.8. Prop.6 Associativity selection - Cumulative window

Considering the description of $[\infty]$ and the proof A.3, we have $S[\infty](t,i) = \{s \in S, \mathcal{B}_S(s) \leq (t,i)\}$. The proof of this property is now easy, using the same method as proof as A.6 with a simplier inequality.■

A.9. Prop.7 Associativity Codd's unary - Streamers

Let's take the case of σ_c and I_S.

Let R be a relation, and c be a valid condition not ot. For simplicity, we note, $S = \sigma_c I_S(R)$

We have the following: $\Psi_{t,i}(s,t) \in \sigma_c I_S(R) \wedge \mathcal{B}_S(\Psi_{t,i}(s,t)) = (t,i)$

$\Leftrightarrow \Psi_{t,i}(s,t) \in I_S(R) \wedge \mathcal{B}_S(\Psi_{t,i}(s,t)) = (t,i) \wedge c(\Psi_{t,i}(s,t))$
$\Leftrightarrow s \in R(t,i) \wedge s \notin R((t,i)^-) \wedge c(\Psi_{t,i}(s,t))$
$\Leftrightarrow s \in R(t,i) \wedge s \notin R((t,i)^-) \wedge c(s)$
$\Leftrightarrow s \in (\sigma_c R)(t,i) \wedge s \notin R((t,i)^-)$
$\Leftrightarrow s \in (\sigma_c R)(t,i) \wedge \neg(s \in R((t,i)^-) \wedge c(s))$
$\Leftrightarrow s \in (\sigma_c R)(t,i) \wedge s \notin \sigma_c R(t,i)$

The equivalence between second and third line is true because c is not on t. The result is now trivial.■

A.10. Cor.3 Selection on stream as a composition

The proof of this property is a direct consequence of the properties 2 and 7.

$$\sigma_c S = \sigma_c I_S(S[\infty]) = I_S(\sigma_c S[\infty])$$

The same approach can be made for I_S and $[B]$ as well as R_S^u and $[B]$.■

A.11. Cor.4 Associativity join - selection

Using prop.6 and prop.7 we can see:
$\sigma_c(S \bowtie_a^d S')$

$$\begin{aligned}
&\equiv \sigma_c I_S(\Pi^{\bowtie}(S[\infty] \underset{\wedge|t'-t|\leq d}{\underset{a}{\bowtie}} (\rho_{t'/t} S'[\infty])) \\
&\equiv I_S(\sigma_c \Pi^{\bowtie}(S[\infty] \underset{\wedge|t'-t|\leq d}{\underset{a}{\bowtie}} (\rho_{t'/t} S'[\infty])) \\
&\equiv I_S(\Pi^{\bowtie}((\sigma_c S[\infty]) \underset{\wedge|t'-t|\leq d}{\underset{a}{\bowtie}} (\rho_{t'/t} S'[\infty])) \\
&\equiv I_S(\Pi^{\bowtie}((\sigma_c S)[\infty] \underset{\wedge|t'-t|\leq d}{\underset{a}{\bowtie}} (\rho_{t'/t} S'[\infty])) \\
&\equiv (\sigma_c S) \bowtie_a^d S'
\end{aligned}$$

Similary we have the same result for the join with the relation.■

A.12. Prop.8 Associativity on product

Considering the product with the lexicographic ordering we will note. $\Phi_{t,i}^{R_1 \times R_2}$.

Let R_1, R_2, R_3 be three relations with no attributes in common.

We want to see if $(R_1 \times R_2) \times R_3 \equiv R_1 \times (R_2 \times R_3)$.

The results on the classical relational algebra shows that the content is equivalent. We now must prove that the ordering is equivalent, hence, prove that the lexicographic ordering is associative.
$(a,(b,c)) \leq (d,(e,f))$

$\Leftrightarrow a < d \vee a = d \wedge (b,c) \leq (e,f)$
$\Leftrightarrow a < d \vee a = d \wedge (b < e \vee b = e \wedge c \leq f)$
$\Leftrightarrow a < d \vee (a = d \wedge b < e) \vee (a = d \wedge b = e \wedge c \leq f)$
$((a,b),c) \leq ((d,e),f)$

$\Leftrightarrow (a,b) < (d,e) \vee (a,b) = (d,e) \wedge c \leq f$
$\Leftrightarrow (a,b) < (d,e) \vee (a = d \wedge b = e \wedge c \leq f)$
$\Leftrightarrow a < d \vee (a = d \wedge b < e) \vee (a = d \wedge b = e \wedge c \leq f)$

Therefore the lexicographic ordering is associative. Thus, the ordering on the left side is equivalent to the one on the right side of the equivalence, hence the result.■

A.13. Cor.5 Associativity on band-join

We want to see if $(S_1 \bowtie^{\infty} S_2) \bowtie^{\infty} S_3 \equiv S_1 \bowtie^{\infty} (S_2 \bowtie^{\infty} S_3)$.

Firstly, we can see that : $I_S(\Pi^{\bowtie} S_1[\infty] \bowtie \rho_{t'/t} S_2[\infty])[\infty]$ is quite similar to $\Pi^{\bowtie} S_1[\infty] \bowtie \rho_{t'/t} S_2[\infty]$ apart from the timestamp (which can be proven to be equal to $\max(s_1(t), s_2(t))$) as this last relation is insert-only. This result is very similar to proof A.5 with timestamp transmission.

Now we can detail the results: $(S_1 \bowtie^{\infty} S_2) \bowtie^{\infty} S_3$

$\equiv I_S(\Pi^{\bowtie} I_S(\Pi^{\bowtie} S_1[\infty] \bowtie \rho_{t'/t} S_2[\infty])[\infty] \bowtie \rho_{t'/t} S_3)$
$\equiv I_S(\Pi^{\bowtie}(\Pi^{\bowtie} S_1[\infty] \bowtie \rho_{t'/t} S_2[\infty]) \bowtie \rho_{t'/t} \bowtie S_3)$
$\equiv I_S(\Pi_{Attr(\ldots)\setminus\{t',t''\}}(S_1[\infty] \bowtie \rho_{t'/t} S_2[\infty]) \bowtie \rho_{t''/t} \bowtie S_3)$
$\equiv I_S(\Pi_{Attr(\ldots)\setminus\{t',t''\}} S_1[\infty] \bowtie \rho_{t'/t} S_2[\infty] \bowtie \rho_{t''/t} \bowtie S_3)$

Therefore, as the relational cartesian product, hence the join, is associative, the last line stands. The band-join is now associative for $d = \infty$.

We can also see why it is impossible to have an associative general band join as the timestamp transmission is not directly effective. We would have a condition on the right side like $|\max(t,t') - t''| \leq d$ which is hardly associative.

The proof is very similar (and easier) as $S_1 \bowtie^0 S_2 = I_S(S_1[\infty] \bowtie_t S_2[\infty])$, therefore we do not need the Π and ρ. ■

B. Transposability

We have presented in section 6 an introduction to the transposability, i.e. the concept of comparing two queries (streams or relation) that started at different timestamp. We detail here a bit further our reflection.

As a recall, we consider a naturally transposable stream S naturally transposable from t_0 to t_1 if : $(\sigma_{t \geq t_1} S, t_0) \equiv (S, t_0)$. Before going any further, we are conscious of many subtle formal problems that exists while manipulating such concepts particulary when stating an equivalence between two queries over time. We have detailed this part a lot more on the Astral's wiki [34] by detailing exactly the equivalence as well as some phenomena that can be observed in the beginning of a query. But we here expose only the global results that we have, particularly on the window operator.

B.1. The general theorem of window transposability

By reusing the notation given in the subsection 6.3. We have the following theorem:

Property 10 (General theorem of window sequence transposability)**.** *Let S be a stream, transposable from t_0 to t_1. i.e. $(\sigma_{t\geq t_1}S, t_0) \equiv (S, t_0)$*

If $K \in \mathbb{N}$ and if $\forall i \in \mathbb{N}$,

$$\begin{cases} \alpha'(i) = \max(B_{t_1}, \alpha(i+K) - D) \\ \beta'(i) = \beta(i+K) - D \end{cases}$$

Then, the following equivalence states:

$$(\sigma_{t\geq t_1}S[\alpha,\beta,r], t_0) \equiv (S[\alpha',\beta',r].t_1)$$

In order to prove this theorem, we need a lemma on the transposability of the τ_S function.

Property 11 (Transposability of τ)**.** *Let S be a stream naturally transposable from t_0 to t_1.*

Let D be a constant equals to the number of tuples that were inside the stream before $(t1, 0)$, which is equals to $\tau^{-1}_{(S,t_0)}((t_1,0)^-) + 1$.

Then, the following equalities states:

$$\begin{array}{rrcl} \forall n \in \mathbb{N}, & \tau_{(S,t_1)}(n) & = & \tau_{(S,t_0)}(n+D) \\ \forall t,i \in T\times\mathbb{N} \geq (t_1,0), & \tau^{-1}_{(S,t_1)}(t,i) & = & \tau^{-1}_{(S,t_0)}(t,i) - D \end{array}$$

Considering that S is naturally transposable, $(S, t_1) = (\sigma_{t\geq t_1}S, t_0)$. Then, it is fairly simple to imply, $\tau_{\sigma_{t\geq t_1}S}(n) = \tau_S(n+D)$ as D is the number of tuples that were inside the stream at t_1. The property on the τ_S^{-1} is a bit more complicated:

$$\begin{array}{rcl} \tau^{-1}_{(S,t_1)}(t,i) & = & \sum_{n=-1}^{+\infty} n\mathbf{1}_{[\tau_{(S,t_1)}(n),\tau_{(S,t_1)}(n+1)[}(t,i) \\ & = & \displaystyle\sum_{n=-1}^{+\infty} n\mathbf{1}_{[\tau_{(S,t_0)}(n+D),\tau_{(S,t_0)}(n+1+D)[}(t,i) \\ & = & \displaystyle\sum_{n=D-1}^{+\infty} (n-D)\mathbf{1}_{[\tau_{(S,t_0)}(n),\tau_{(S,t_0)}(n+1)[}(t,i) \\ & = & \tau^{-1}_{(S,t_0)}(t,i) - D \\ & & \displaystyle - \sum_{n=-1}^{D-2} (n-D)\mathbf{1}_{[\tau_{(S,t_0)}(n),\tau_{(S,t_0)}(n+1)[}(t,i) \end{array}$$

As we take $(t,i) \geq (t_1,0)$, and $\tau_{(S,t_0)}(D-1) \leq (t_1,0)$, the last sum is always null. Hence the property. ■

Now we can prove the general theorem of window sequence transposability.

As a beginning, it is trivial to see that we must have the same rate for update frequency reasons.

Let γ and γ' be the delaying function implied by the given descriptions.

- In the temporal case

  Let (t,i) such that $\gamma'(t,i) \geq 0$,

  The condition over the stream on the window sequence operator on the first query will be: $(\alpha(\gamma(t,i)),0) \leq \mathcal{B}_S(s) \leq (\beta(\gamma(t,i)),i)$

  As the input stream is naturally transposable, we have $(\sigma_{t\geq t_1}S, t_0) = (S, t_1)$, therefore we restrict the precedent condition with $t \geq t_1$ $(\max(\alpha(\gamma(t,i)),t_1),0) \leq \mathcal{B}_S(s) \leq (\beta(\gamma(t,i)),i)$

  We want that the condition implied by $[\alpha',\beta',r]$ verify this same condition. We can firstly see that,

$$\begin{array}{rcll} \gamma'(t,i) & = & \left\lfloor \frac{t-\beta'(0)}{r} \right\rfloor & \\ & = & \left\lfloor \frac{t-\beta(0)+\beta(0)-\beta'(0)}{r} \right\rfloor & \\ & = & \left\lfloor \frac{t-\beta(0)}{r} - K \right\rfloor & \\ & = & \gamma(t,i) - K & \text{as } K \in \mathbb{N} \end{array}$$

  We now have in the second query, the following conditions:

$$\begin{array}{rcl} (\alpha'(\gamma'(t,i)),0) & \leq \mathcal{B}_S(s) \leq & (\beta'(\gamma'(t,i)),i) \\ (\alpha'(\gamma(t,i)-K),0) & \leq \mathcal{B}_S(s) \leq & (\beta'(\gamma(t,i)-K),i) \end{array}$$

  Considering the conditions on α and β, the conditions over the first stream and the second are exactly equivalent. We do have proven the equality for any batch on the stabilisation period.

- In the positional case

  This case is far more complex to prove. As in the temporal case, we want to verify that the condition are equivalent.

  Let $j \in \mathbb{N}$, such that $j \geq K$,

$$\begin{array}{rcl} \alpha'(j) & = & \max(0, \alpha(j+K) - D) \\ \alpha'(j) + D & = & \max(D, \alpha(j+K)) \\ \alpha'(j-K) + D & = & \max(D, \alpha(j)) \\ \tau_{(S,t_0)}(\alpha'(j-K)+D) & = & \max(\tau_{(S,t_0)}(D), \tau_{(S,t_0)}(\alpha(j))) \end{array}$$

  The last statement is true because τ is growing non-strictly due to the coherency between position and timestamp hypothesis.

  Thanks to the property 11 on the transposability of τ, we have,

$$\tau_{(S,t_1)}(\alpha'(j-K)) = \max((t_1,0), \tau_{(S,t_0)}(\alpha(j)))$$

  Now we try to verify the same condition over γ' that we have in the temporal case.

$$\begin{array}{rcll} \gamma'(t,i) & = & \left\lfloor \frac{\tau_{(S,t_1)}(t)-\beta'(0)}{r} \right\rfloor & \\ & = & \left\lfloor \frac{\tau_{(S,t_0)}(t)-D-\beta'(0)}{r} \right\rfloor & \\ & = & \left\lfloor \frac{\tau_{(S,t_0)}(t)-D-\beta(0)+\beta(0)-\beta'(0)}{r} \right\rfloor & \\ & = & \left\lfloor \frac{\tau_{(S,t_0)}(t)-\beta(0)}{r} - K \right\rfloor & \\ & = & \left\lfloor \frac{\tau_{(S,t_0)}(t)-\beta(0)}{r} \right\rfloor - K & \text{as } K \in \mathbb{N} \\ & = & \gamma(t,i) - K & \end{array}$$

We can now conclude by saying that: $\forall t, i \geq \gamma'(t,i) \geq 0, \tau_{(S,t_1)}(\alpha'(\gamma'(t,i)) = \max(\tau_{(S,t_0)}(\alpha(\gamma(t,i))), (t_1, 0))$

The demonstration is exactly the same for β'.■

B.2. Prop.9 Linear Window Transposing

Considering the prop.10 we try to respect the different conditions :

- As $K = \frac{\beta'(0)-\beta(0)+D}{r} = \frac{d'-d+D}{r}$, the first condition is natural.
- As $\beta'(i) = \beta(i+K)-D \Leftrightarrow c'i+d' = ci+cK+d-D$. This condition is true if and only if $c' = c$. Therefore we have, $cK = d'-d+D$, if $K \neq 0$, then $c = r$ (else no condition is necessary).
- As $\alpha'(i) = \max(B_{t_1}, \alpha(i+K)-D) \Leftrightarrow \max(a'i+b', B_{t_1}) = \max(B_{t_1}, \max(B_{t_0}, ai+aK+b-D)) = \max(B_{t_1}, ai+aK+b-D)$. Considering $a \neq 0$, and i enough big to go beyond B_{t_1}, we can imply that $a = a'$ (this last equality is also trivially true for $a = 0$). We also directly imply that $b' = aK+b-D$. In the case of $a = 0$, no further simplification can be made.

B.3. Cor.6 Linear Natural Transposing

The demonstration uses directly the results from the linear case.

We have $d+B_{t_1}-d-B_{t_0}+D \in r\mathbb{N} \Leftrightarrow B_{t_1}-B_{t_0}+D \in r\mathbb{N}$. In the temporal case, $t_1-t_0 \in r\mathbb{N}$ and in the positional case $\tau^{-1}_{(S,t_0)}(t_1) \in r\mathbb{N}$ (considering directly the definition of B and D).

We do have a $K \neq 0$ as $t_1 \neq t_0$ by definition (or the same in the positional case which is not interesting). Then $c = r$.

Let's consider $a \neq 0$, then $B_{t_1} = B_{t_0} + aK - D \Leftrightarrow aK = B_{t_1} - B_{t_0} + D = rK$. Therefore, $a = r$.

Let's consider $a = 0$, then $\max(B_{t_1}, b + B_{t_1}) = \max(B_{t_1}, b + B_{t_0} - D)$.

- If $b < 0$, then $b + B_{t_1} = \max(B_{t_1}, b + B_{t_0} - D) \Leftrightarrow b = \max(0, b - rK)$, then $b = 0$ which is absurd.
- If $b \geq 0$, then $B_{t_1} = \max(B_{t_1}, b + B_{t_0} - D) \Leftrightarrow 0 = \max(0, b - rK)$ which is true for any $b \geq 0$


L I G